ON USING INDUCTIVE BIASES FOR DESIGNING DEEP LEARNING ARCHITECTURES

A Dissertation
Presented to
The Academic Faculty

By

Harsh Shrivastava

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Machine Learning

Georgia Institute of Technology

May 2021

Copyright © Harsh Shrivastava 2021

ON USING INDUCTIVE BIASES FOR DESIGNING DEEP LEARNING ARCHITECTURES

Approved by:

Dr. Srinivas Aluru, Advisor
School of Computational Science and Engineering
Georgia Institute of Technology

Dr. Le Song
School of Computational Science and Engineering
Georgia Institute of Technology

Dr. Xiuwei Zhang
School of Computational Science and Engineering
Georgia Institute of Technology

Dr. B. Aditya Prakash
School of Computational Science and Engineering
Georgia Institute of Technology

Dr. Ashok Goel
School of Interactive Computing
Georgia Institute of Technology

Date Approved: December 9, 2020

If you can't solve a problem, then there is an easier problem you can't solve: find it!

George Pólya

To my mother for nurturing my dreams. To my father for instilling scientific curiosity in me from an early age and channeling my excitement towards critical thinking.

ACKNOWLEDGEMENTS

I will start by expressing my immense gratitude towards my advisor Dr. Srinivas Aluru. Thank you professor for your constant support, encouraging comments and constructive criticism throughout my PhD years. Thank you for your timely guidance, for facilitating many conference visits and giving me a freehand to pursue my research interests.

To my thesis committee members, Dr. Le Song, Dr. Xiuwei Zhang, Dr. B. Aditya Prakash and Dr. Ashok Goel for your advice, comments and valuable feedback in making this dissertation possible. I will like to further thank Dr. Song for mentoring and guiding me, especially with the machine learning components of this dissertation. I am much obliged to you for your advice, support and patience throughout my PhD.

I will like to thank my current labmates Ankit, Neda, Shruti, Haowen for being an amazing bunch of researchers, for their help and more importantly for tolerating my bad puns. I consider my good fortune to have a very energetic and eclectic friend circle at Georgia Tech: Patrick F., Srinivas, Patrick L., Rahul, Sriram, Tony, Saminda, Rakshit, Michael 'mikey', Chunxing, Wafa, Elias, Lanssie, Rob, Chirag, Majid, Jiajia, Apo, Xiaojing, Chris, Shivesh, Xinshi, Kerianne and the list goes on. I learned a lot from you guys, academic stuff as well as important life skills. Hoping to cross paths with you all again in near future.

A heart-felt thanks to the academic staff of CSE and ML department. Kristen, Allie, Stephanie, Arlene, Nirvana, Shkina, Carolyn, Anna and others. Thank you for your patience, timely responses and for facilitating a healthy and collaborative environment.

I had a good fortune to do internships at reputed industry research labs during the course of my PhD. I am very grateful to my internship mentors and the friendships fostered during those times. I am thankful to the various groups I have been a part of during my stay in Atlanta: Badminton club, Squash buddies, Pool group, Tennis group, visiting art shows etc. I will always cherish those memories and they also provided the necessary break from

research which in-fact helped improve my productivity.

Despite being in a different continent, my wingmates from undergrad have been in constant touch throughout and I have always enjoyed our uninhibited conversations. I am very grateful to my parents, my younger brother Gaurav and my relatives for their continued support and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Recent advancements in field of Artificial Intelligence, especially in the field of Deep Learning (DL), have paved way for new and improved solutions to complex problems occurring in almost all domains. Often we have some prior knowledge and beliefs of the underlying system of the problem at-hand which we want to capture in the corresponding deep learning architectures. Sometimes, it is not clear on how to include our prior beliefs into the traditionally recommended deep architectures like Recurrent neural networks, Convolutional neural networks, Variational Autoencoders and others. Often the post-hoc techniques of modifying these architectures are not straightforward and provide little performance gain.

There have been efforts on developing domain specific architectures but those techniques are generally not transferable to other domains. We ask the question that can we come up with generic and intuitive techniques to design deep learning architectures that takes our prior knowledge of the system as an inductive bias?

In this dissertation, we develop two novel approaches towards this end. The first one called 'Cooperative Neural Networks' can incorporate the inductive bias from the underlying probabilistic graphical model representation of the domain. The second one called problem dependent 'Unrolled Algorithms' parameterizes the recurrent structure of unrolling the iterations of an optimization algorithm for the objective function defining the problem. We found that the neural network architectures obtained from our approaches typically end up with very fewer learnable parameters and provide considerable improvement in run-time compared to other deep learning methods. We have successfully applied our techniques to solve Natural Language processing related tasks, doing sparse graph recovery and computational biology problems like doing gene regulatory network inference.

Firstly, we introduce the Cooperative Neural Networks approach which is a new theoretical approach for implementing learning systems that can exploit both prior insights about the independence structure of the problem domain and the universal approximation

capability of the deep neural networks. Specifically, we develop CoNN-sLDA model for the document classification task. We use the popular Latent Dirichlet Allocation graphical model as the inductive bias for the CoNN-sLDA model. We demonstrate a 23% reduction in error on the challenging MultiSent data set compared to state-of-the-art and also derived ways to make the learned representations more interpretable.

Secondly, we elucidate the idea of using problem dependent 'Unrolled Algorithms' for the sparse graph recovery task. We propose a deep learning architecture, `GLAD`, which uses an Alternating Minimization algorithm as our model inductive bias and learns the model parameters via supervised learning. We show that `GLAD` learns a very compact and effective model for recovering sparse graphs from data. We do an extensive theoretical analysis that strengthen our claims for using similar approaches for other problems as well.

Finally, we further build up on the proposed 'Unrolled Algorithm' technique for a challenging real world computational biology problem. To this end, we design `GRNUlar`, a novel deep learning framework for supervised learning of gene regulatory networks (GRNs) from single cell RNA-Sequencing data. Our framework incorporates two intertwined models. We first leverage the expressive ability of neural networks to capture complex dependencies between transcription factors and the corresponding genes they regulate, by developing a multi-task learning framework. Then, in order to capture sparsity of GRNs observed in the real world, we design an unrolled algorithm technique for our framework. Our deep architecture requires supervision for training, for which we repurpose existing synthetic data simulators that generate scRNA-Seq data guided by an underlying GRN. Experimental results demonstrate `GRNUlar` outperforms state-of-the-art methods on both synthetic and real datasets. Our work also demonstrates the novel and successful use of expression data simulators for supervised learning of GRN inference.

CHAPTER 1
INTRODUCTION AND MOTIVATION

The field of Artificial Intelligence is progressing rapidly, especially the developments in Deep Learning (DL) research, that have paved way for new and improved solutions to complex problems occurring in almost all domains [1]. Deep learning based architectures like Deep neural networks, Convolutional neural networks, Variational Autoencoders, Generative Adversarial Networks, Recurrent neural networks etc. have been successful in establishing state-of-the-art results for numerous complex problems in various fields [1]. Although, these architectures are widely used for many domains, it is often not straightforward to include or capture any prior knowledge of a domain into their deep architecture in an organic fashion.

The prior knowledge of a domain can be described in multiple ways. Probabilistic graphical models (PGMs) are one such powerful framework which are widely used to represent systems ranging from healthcare, finance, social media, computational biology and so on [2]. Another way, though slightly subtle, is to make use of an existing optimization algorithm for the objective function relevant to the problem at-hand. Prior approaches for designing deep learning architectures have at times used these prior knowledge for a particular problem, but their technique was limited to that problem and cannot be easily extended to designing architectures for problems in different domains.

Thus, the motivation of this dissertation is to come up with generic techniques that can be used to design deep learning architectures which can leverage the domain specific inductive bias and in-turn perform better than their traditional counterparts. This thesis provides 2 such novel techniques, namely 'Cooperative Neural Networks' and problem dependent 'Unrolled Algorithms', for designing deep learning architectures. We applied these techniques to problems related to Natural Language processing, document sentiment analysis, doing sparse graph recovery for gene regulatory networks. Throughout our experiments, we find

that our deep architectures require considerably less amount of data for training and runs significantly faster than other competing deep architectures. This is mainly attributed to the considerably less number of learnable neural network parameters needed for our techniques.

We believe that this dissertation can give rise to a new paradigm of designing deep learning architectures, one that can reflect our belief of the underlying system of the domain under consideration.

1.1 Preliminaries & Related works

This section gives a brief survey of the basic concepts that will be helpful to follow the consequent chapters. Each chapter also contain their specific literature survey as well. Since, the focus of this thesis is on developing DL architectures for problems in different domains, this section also provides some basic domain specific references that will be useful for in-depth understanding.

- Deep learning and neural networks background: Deep learning is a fast developing field and the research is progressing with a tremendous pace. It is indeed a difficult task to keep up with the latest developments in the field but there are certain widely accepted concepts that will be useful in following this thesis. Deep neural networks (DNNs) are able to learn expressive class of functions [1] and this viewpoint of DNNs is often leveraged in this thesis. Whenever we come across opportunities to parameterize certain problem dependent hyperparameters, especially in Chapters 3&4, we make use of DNNs as they can be adapted to the problem at-hand using data driven training. Basic knowledge of Recurrent neural networks, LSTMs [3, 4] will be helpful as it is critical to have a good understanding of them to be able to follow the new deep and recurrent architectures proposed in this work. A basic tutorial for introduction to Convolutional neural networks can be found in [5]. Understanding of generative deep learning models like Variational Autoencoders [6] and Generative adversarial networks [7] will help the reader to grasp the underlying concepts of Chapter 2 as we connect deep learning with probabilistic graphical

models.

- Probabilistic graphical models: Understanding of the functioning of Probabilistic graphical models(PGMs) [2] like Bayesian Networks and Markov Networks are key to follow the work presented in this thesis. Though, every chapter motivates and elucidates the PGMs under consideration, it is a helpful exercise to follow the seminal work given in [8]. This work explains the Latent Dirichlet Allocation graphical model which is widely used for topic modeling of a document corpus. Pay special attention to the variational inference concept as we will provide a follow up of that technique in Chapter 2 by making use of Hilbert space embeddings of distributions [9]. Since, Chapter 2 gives a general technique of using the inductive bias provided by an underlying PGM to design a domain specific deep architecture, it will be helpful to understand the PGMs prevalent and widely used for various applications. PGMs are widely used to represent and infer systems in healthcare [10], finance [11, 12], social networks [13] and computational biology [14, 15]. A good insight into these PGMs will help the reader to adapt the generic techniques provided in this thesis to the problem of their interest.
- Optimization literature survey: Chapters 3&4 require a bit in-depth knowledge of convex and non-linear optimization techniques [16, 17]. Especially, a good primer on Alternating Direction method of multipliers (ADMM) [18], Iterative Shrinkage Thresholding algorithm (ISTA) [19] and methods combining deep neural networks based on unfolding of these optimization algorithms [20, 21] will be helpful. Chapter 3 makes extensive use of the optimization of the graphical lasso objective [22].
- Computational biology literature survey: Chapter 4 is our attempt to apply our newly developed techniques to provide a solution to a very important problem of doing Gene regulatory network inference from the single cell RNA sequencing data. A recent survey on the problem of Gene regulatory network inference from single-cell transcriptomic data and the existing methods can be found in [23]. An insight of the synthetic data simulators for generating gene expression data will be required as well [24, 25]. For the readers

with background in computational biology, a more detailed and comprehensive literature survey is provided in the Chapter 4 itself.

1.2 Contributions & Structure of this dissertation

In this dissertation, we present two different paradigms of deep learning architecture designs which are based on utilizing the inductive biases provided by our prior knowledge of the problem domain. These two techniques are called 'Cooperative Neural Networks' and problem dependent 'Unrolled algorithms'. These are generic techniques and can be used to design deep architectures for problems from various domains. These architectures usually require supervision for training for which we provide innovative solutions. We also show that the amount of supervision data needed is significantly less than their traditional counterparts. In this thesis, we evaluate our approaches on Natural Language Processing tasks like document sentiment analysis, on the important problem of doing sparse graph recovery which can in-turn help with interpretability of predictions and on problems of significance in computational biology like doing Gene regulatory network reconstruction from single-cell RNA sequencing data.

In Chapter 2, we propose a new approach, called cooperative neural networks (CoNN), which uses a set of cooperatively trained neural networks to capture latent representations that exploit prior given independence structure. The model is more flexible than traditional graphical models based on exponential family distributions, but incorporates more domain specific prior structure than traditional deep networks or variational autoencoders. The framework is very general and can be used to exploit the independence structure of any graphical model. We illustrate the technique by showing that we can transfer the independence structure of the popular Latent Dirichlet Allocation (LDA) model to a cooperative neural network, CoNN-sLDA. Empirical evaluation of CoNN-sLDA on supervised text classification tasks demonstrates that the theoretical advantages of prior independence structure can be realized in practice - we demonstrate a 23% reduction in error on the challenging

MultiSent data set compared to state-of-the-art. We further develop this approach to be more interpretable and also devised ways to use the intermediate representations learned by the Cooperative Neural Networks.

Recovering sparse conditional independence graphs from data is a fundamental problem in machine learning with wide applications. A popular formulation of the problem is an ℓ_1 regularized maximum likelihood estimation. Many convex optimization algorithms have been designed to solve this formulation to recover the graph structure. Recently, there is a surge of interest to learn algorithms directly based on data, and in this case, learn to map empirical covariance to the sparse precision matrix. However, it is a challenging task in this case, since the symmetric positive definiteness (SPD) and sparsity of the matrix are not easy to enforce in learned algorithms, and a direct mapping from data to precision matrix may contain many parameters. In Chapter 3, we propose a deep learning architecture, `GLAD`, which uses an Alternating Minimization (AM) algorithm as our model inductive bias, and learns the model parameters via supervised learning. We show that `GLAD` learns a very compact and effective model for recovering sparse graphs from data. We do extensive theoretical analysis to support our claims and also demonstrate that in the case of sparse graph recovery, data-driven learning can also improve the sample complexity.

In Chapter 4, we propose `GRNUlar`, a novel deep learning framework for supervised learning of gene regulatory networks (GRNs) from single cell RNA-Sequencing data. Our framework incorporates two intertwined models. Firstly, we leverage the expressive ability of neural networks to capture complex dependencies between transcription factors and the corresponding genes they regulate, by developing a multi-task learning framework. Secondly, in order to capture sparsity of GRNs observed in the real world, we design an unrolled algorithm technique for our framework. Our deep architecture requires supervision for training, for which we repurpose existing synthetic data simulators that generate scRNA-Seq data guided by an underlying GRN. Experimental results demonstrate `GRNUlar` outperforms state-of-the-art methods on both synthetic and real datasets. Our work also

demonstrates the novel and successful use of expression data simulators for supervised learning of GRN inference.

Chapter 5 gives an overall conclusion of this thesis by summarizing the contribution and overall research impact of this research work. The original work presented in this dissertation is published in the following research papers:

- Shrivastava, Harsh, et al. 'Cooperative neural networks (CoNN): Exploiting prior independence structure for improved classification.' Advances in Neural Information Processing Systems (Neurips) 2018.
- Shrivastava, Harsh, et al. '`GLAD`: Learning sparse graph recovery,' in International Conference on Learning Representations (ICLR) 2020
- Shrivastava, Harsh, et al. 'GRNUlar: Gene Regulatory Network reconstruction using Unrolled algorithm from Single Cell RNA-Sequencing data.' (bioRxiv) 2020.
- Shrivastava, Harsh, et al. 'An unrolled deep learning framework for single cell gene regulatory networks.' in International Conference on Research in Computational Molecular Biology (RECOMB) 2021.

CHAPTER 2

COOPERATIVE NEURAL NETWORKS (CONN): EXPLOITING PRIOR INDEPENDENCE STRUCTURE FOR IMPROVED CLASSIFICATION

2.1 Introduction

Neural networks offer a low-bias solution for learning complex concepts such as the linguistic knowledge required to separate documents into thematically related classes. However, neural networks typically start with a fairly generic structure, with each level comprising a number of functionally equivalent neurons connected to other layers by identical, repetitive connections. Any structure present in the problem domain must be learned from training examples and encoded as weights. In practice, some domain structure is often known ahead of time; in such cases, it is desirable to pre-design a network with this domain structure in mind. In this paper, we present an approach that allows incorporating certain kinds of independence structure into a new kind of neural learning machine.

The proposed approach is called "Cooperative Neural Networks" (CoNN). This approach works by constructing a set of neural networks, each trained to output an embedding of a probability distribution. The networks are iteratively updated so that each embedding is consistent with the embeddings of the other networks and with the training data. Like probabilistic graphical models, the representation is factored into components that are independent. Unlike probabilistic graphical models, which are limited to tractable conditional probability distributions (e.g., exponential family), CoNNs can exploit powerful generic distributions represented by non-linear neural networks. The resulting approach allows us to create models that can exploit both known independence structure as well as the expressive powers of neural networks to improve accuracy over competing approaches.

We illustrate the general approach of cooperative neural networks by showing how

one can transfer the independence structure from the popular Latent Dirichlet Allocation (LDA) model [8] to a set of cooperative neural networks. We call the resultant model CoNN-sLDA. Cooperative neural networks are different from feed forward networks as they use back-propagation to enforce consistency across variables within the latent representation. CoNN-sLDA improves over LDA as it admits more complex distributions for document topics and better generalization over word distributions. CoNN-sLDA is also better than a generic neural network classifier as the factored representation forces a consistent latent feature representation that has a natural relationship between topics, words and documents. We demonstrate empirically that the theoretical advantages of cooperative neural networks are realized in practice by showing that our CoNN-sLDA model beats both probabilistic and neural network-based state-of-the-art alternatives. We emphasize that although our example is based on LDA, the CoNN approach is general and can be used with other graphical models, as well as other sources of independence structure (for example, physics- or biology-based constraints).

2.2 Related Work

Text classification has a long history beginning with the use of support vector machines on text features [26]. More sophisticated approaches integrated unsupervised feature generation and classification in models such as sLDA [27, 28] and discriminative LDA (discLDA) [29] and a maximum margin based combination [30].

One limitation of LDA-based models is that they pick topic distributions from a Dirichlet distribution and cannot represent the joint probability of topics in a document (i.e., hollywood celebrities, politics and business are all popular categories, but politics and business appear together more often than their independent probabilities would predict). Models such as pachinko allocation [31] attempt to address this with complex tree structured priors. Another limitation of LDA stems from the fact that word topics and words themselves are selected from categorical distributions. These admit arbitrary empirical distributions over

tokens, but don't generalize what they learn. Learning about the topic for the token "happy" tells us nothing about the token "joyful".

There have been many generative deep learning models such as Deep Boltzmann Machines [32], NADE [33, 34], variational auto-encoders (VAEs) [35] and variations [36], GANs[37] and other deep generative networks [38, 39, 40, 41] which can capture complex joint distributions of words in documents and surpass the performance of LDA. These techniques have proven to be good generative models. However, as purely generative models, they need a separate classifier to assign documents to classes. As a result, they are not trained end-to-end for the actual discriminative task that needs to be performed. Therefore, the resulting representation that is learned does not incorporate any problem-specific structure, leading to limited classification performance. Supervised convolutional networks have been applied to text classification [42] but are limited to small fixed inputs and still require significant data to get high accuracy. Recurrent networks have also been used to handle open ended text [43]. A supervised approach for LDA with DNN was developed by [44, 45] using end-to-end learning for LDA by using Mirror-Descent back propagation over a deep architecture called BP-sLDA. To achieve better classification, they have to increase the number of layers of their model, which results in higher model complexity, thereby limiting the capability of their model to scale. In summary, there are still significant challenges to creating expressive, but efficiently trainable and computationally tractable models.

In the face of limited data, regularization techniques are an important way of trying to reduce overfitting in neural approaches. The use of pretrained layers for networks is a key regularization strategy; however, training industrial applications with domain specific language and tasks remains challenging. For instance, classification of field problem reports must handle content with arcane technical jargon, abbreviations and phrasing and be able to output task specific categories.

Techniques such as L2 normalization of weights and random drop-out [46] of neurons during training are now widely used but provide little problem specific advantage. Bayesian

neural networks with distributions have been proposed, but independent distributions over weights result in network weight means where the variance must be controlled fairly closely so that relative relationship of weights produces the desired computation. Variational auto-encoders explicitly enable probability distributions and can therefore be integrated over, but are still largely undifferentiated structure of identical units. They don't provide a lot of prior structure to assist with limited data.

Recently there has been work incorporating other kinds of domain inspired structure into networks such Spatial transformer networks [47], capsule networks [48] and natural image priors [49].

2.3 Deriving Cooperative Neural Networks for LDA

Application of our approach proceeds in several distinct steps. First, we define the independence structure for the problem. In our supervised text classification example, we incorporate structure from Latent Dirichlet Allocation (LDA) by choosing to factor the distribution over document texts into document topic probabilities and word topic probabilities. This structure naturally enforces the idea that there are topics that are common across all documents and that documents express a mixture of these topics independently through word choices. Second, a set of inference equations is derived from the independence structure. Next, the probability distributions involved in the variational approximation, as well as the inference equations, are mapped into a Hilbert space to reduce limitations on their functional form. Finally, these mapped Hilbert-space equations are approximated by a set of neural networks (one for each constraint), and inference in the Hilbert space is performed by iterating these networks. We call the combination of Cooperative Neural Networks and LDA as Cooperative Neural Network supervised Latent Dirichlet Allocation, or 'CoNN-sLDA'. These steps are elaborated in the following sections.

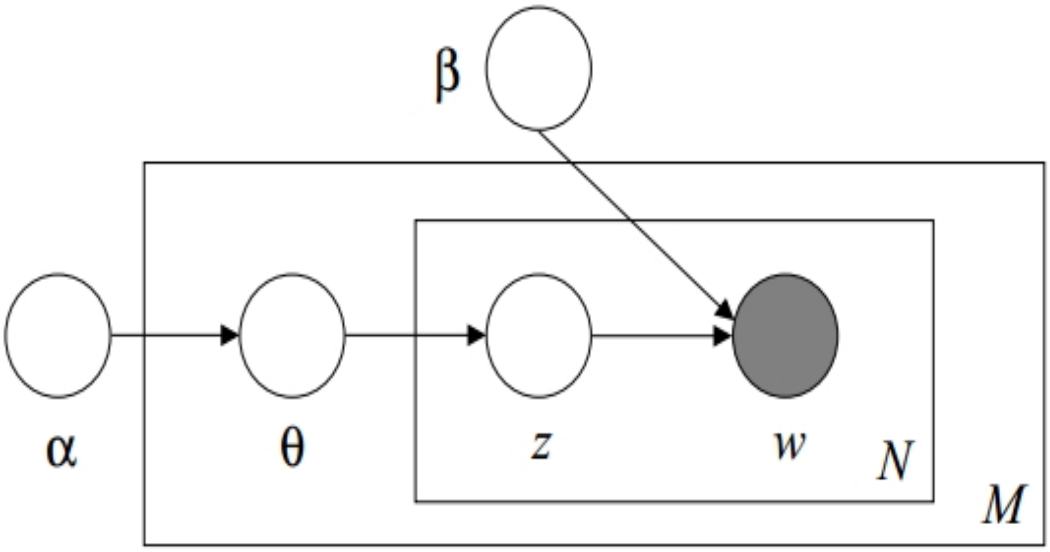

Figure 2.1: The plate model of the Latent Dirichlet Allocation. LDA summarizes the content of each document m in M as a topic distribution θ_m. Each word $w_{m,n}$ in N_m has topic $z_{m,n}$ drawn from θ_m.

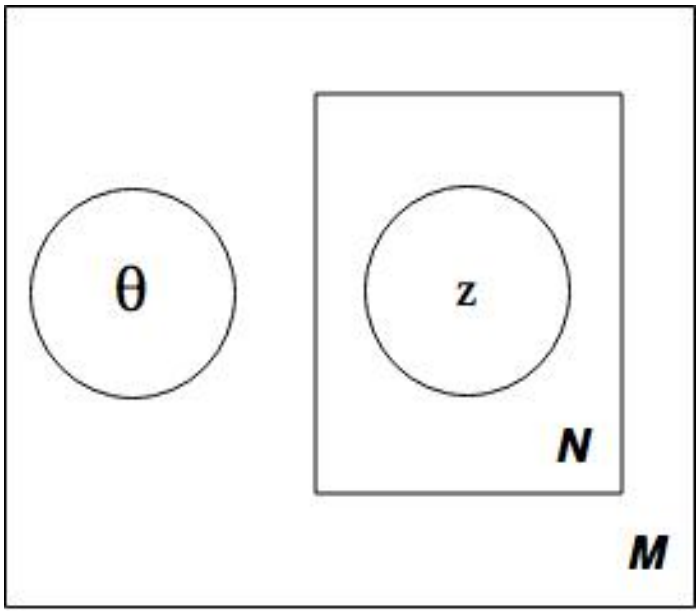

Figure 2.2: Plate model representing the approximate LDA distribution. It represents an approximation to the same corpus of M documents having N words. Variational LDA approximates the posterior topic distribution θ_m and word topic $z_{m,n}$ with independent distributions.

2.3.1 LDA model

Here, we use the same notation and the same plate diagram (Fig. 2.1) as in the original LDA description [8]. Let K be the number of topics, N be the number of words in a document, V be the vocabulary size over the whole corpus, and M be the number of documents in the corpus. Given the prior over topics α and topic word distributions β, the joint distribution over the latent topic structure θ, word topic assignments $\mathbf{z}$, and observed words in documents $\mathbf{w}$ is given by:

$$p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta) = p(\theta|\alpha)\prod_{i=1}^{N} p(z_i|\theta)p(w_i|z_i, \beta) \tag{2.1}$$

2.3.2 Variational approximation to LDA

Inference in LDA requires estimating the distribution over θ and $\mathbf{z}$. Using the Bayes rule, this posterior can be written as follows:

$$p(\theta, \mathbf{z}|\mathbf{w}, \alpha, \beta) = \frac{p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta)}{p(\mathbf{w}|\alpha, \beta)} \tag{2.2}$$

Unfortunately, directly marginalizing out θ in the original model is intractable. Variational approximation of $p(\theta, \mathbf{z})$ is a common work-around. To perform variational approximation, we approximate this LDA posterior with the Probabilistic Graphical Model (PGM) shown in Fig. 2.2. The joint distribution for the approximate PGM is given by:

$$q(\theta, \mathbf{z}) = q(\theta)\prod_{i=1}^{N} q_i(z_i) \tag{2.3}$$

We want to tune the approximate distribution to resemble the true posterior as much as possible. To this end, we minimize the KL divergence between the two distributions. Alternatively, this can be seen as minimizing the variational free energy of the Mean-Field

inference algorithm [50]:

$$\min_{\{q\}} \{D_{KL}(\ q(\theta, \mathbf{z})\ ||\ p(\theta, \mathbf{z}|\mathbf{w}, \alpha, \beta)\ )\} \tag{2.4}$$

To solve this minimization problem, we derive a set of fixed-point equations in following Section 2.3.3.

2.3.3 Derivation of fixed point equations

Inference in LDA requires estimating the distribution over θ and $\mathbf{z}$. Using the Bayes rule, this posterior can be written as follows:

$$p(\theta, \mathbf{z}|\mathbf{w}, \alpha, \beta) = \frac{p(\theta, z, w|\alpha, \beta)}{p(w|\alpha, \beta)} \tag{2.5}$$

To perform variational approximation, we approximate this LDA posterior with the PGM as shown in Figure 2.2.

The joint distribution for the approximate PGM is given by:

$$q(\theta, z) = q(\theta)\prod_{i=1}^{N} q_i(z_i) \tag{2.6}$$

We want to tune the approximate distribution to resemble the true posterior as much as possible. To this end, we minimize the KL divergence between the two distributions. Alternatively, this can be seen as minimizing the variational free energy of the Mean-Field inference algorithm [50]:

$$\min_{\{q\}} \{D_{KL}(\ q(\theta, z)\ ||\ p(\theta, z|w, \alpha, \beta)\ )\} \tag{2.7}$$

Substituting the expression for KL-divergence, we get

$$\min_{\{q\}} \int_{\theta} \int \cdots \int_{\{z_i\}} q(\theta, z) \ \log \frac{q(\theta, z)}{p(\theta, z|w, \alpha, \beta)} \ d\theta \ \{dz_i\} \tag{2.8}$$

Using the Bayes formulation given in equation(2.2) and observing that $p(w|\alpha, \beta)$ is a constant, we can write

$$\min_{\{q\}} \int_{\theta} \int \cdots \int_{\{z_i\}} q(\theta, z) \ [\ \log q(\theta, z) - \log p(\theta, z, w|\alpha, \beta) \ ] \ d\theta \ \{dz_i\} \tag{2.9}$$

Substituting the probability densities given in equations (2.1) and (2.3), the following minimization expression is obtained:

$$\begin{aligned} \min_{\{q\}} \int_{\theta} \int_{\{z_i\}} \left\{ q(\theta) \prod_{i=1}^{N} q_i(z_i) \right\} \Bigg\{ \log \left(q(\theta) \prod_{i=1}^{N} q_i(z_i) \right) \\ - \log \left(p(\theta|\alpha) \prod_{i=1}^{N} p(z_i|\theta) p(w_i|z_i, \beta) \right) \Bigg\} \ d\theta \ \{dz_i\} \end{aligned} \tag{2.10}$$

Pulling logarithms inwards we can convert products to summations. We then move integrals inward. In some cases, integrals add up to 1 (*e.g.*, $\int_{\theta} q(\theta) \ \ d\theta = 1$). In some cases, inner sums can be pulled outwards. The result consists of simple integrals:

$$\begin{aligned} \min_{\{q\}} \Big\{ \ \ & \int_{\theta} q(\theta) \log q(\theta) \ d\theta + \sum_{i=1}^{N} \int_{z_i} q_i(z_i) \log q_i(z_i) \ dz_i - \int_{\theta} q(\theta) \log p(\theta|\alpha) \ d\theta \\ & - \sum_{i=1}^{N} \iint_{\theta, z_i} q(\theta) q_i(z_i) \log p(z_i|\theta) \ d\theta \ dz_i - \sum_{i=1}^{N} \int_{z_i} q_i(z_i) \log p(w_i|z_i, \beta) \ dz_i \ \ \Big\} \end{aligned} \tag{2.11}$$

We denote the expression given in equation(2.11) by $\min_{\{q\}}(L)$. To minimize the functional equation given by L, we take the functional derivatives of L with respect to $q(\theta)$ and $q_i(z_i)$ and equate them to zero.

Solving for $\left(\frac{\delta L}{\delta q(\theta)} = 0\right)$, we get the first fixed point equation:

$$\log q(\theta) = \log p(\theta|\alpha) + \sum_{i=1}^{N} \int_{z_i} q_i(z_i) \log p(z_i|\theta)\ dz_i - 1 \tag{2.12}$$

Similarly, solving for $\frac{\delta L}{\delta q_i(z_i)} = 0$, we get the second fixed point equation:

$$\log q_i(z_i) = \log p(w_i|z_i, \beta) + \int_{\theta} q(\theta) \log p(z_i|\theta) d\theta - 1 \tag{2.13}$$

Note that this derivation is different from the classical variational approximation derivations, where the EM algorithm is eventually used to iteratively approximate the posterior.

We now observe these fixed-point equations that can be expressed concisely as

$$\log q(\theta) = \log p(\theta|\alpha) + \sum_{i=1}^{N} \int_{z_i} q_i(z_i) \log p(z_i|\theta)\ dz_i - 1 \tag{2.14}$$

$$\log q_i(z_i) = \log p(w_i|z_i, \beta) + \int_{\theta} q(\theta) \log p(z_i|\theta) d\theta - 1 \tag{2.15}$$

This set of equations is difficult to solve analytically. In addition, even if it was possible to solve them analytically, they are still subject to the limitations of the original graphical models, such as the need to use exponential family distributions and conjugate priors for tractability.

Therefore, the next step in the proposed method is to map the probability distributions and the corresponding fixed-point equations into a Hilbert space, where some of these limitations can be relaxed. Section 2.3.4 gives a general overview of Hilbert space embeddings, and section 2.3.5 derives the corresponding equations for our model.

2.3.4 Hilbert Space Embeddings of Distributions

We follow the notations and procedure defined in [51] for parameterizing Hilbert spaces. By definition, the Hilbert Space embeddings of probability distributions are mappings of these distributions into potentially *infinite* -dimensional feature spaces. [9]. For any given distribution $p(X)$ and a feature map $\phi(x)$, the embedding $\mu_X : \mathcal{P} \rightarrow \mathcal{F}$ is defined as:

$$\mu_X \ := \ E_X[\phi(X)] \ = \ \int_{\mathcal{X}} \phi(x)p(x)dx \tag{2.16}$$

For some choice of feature map ϕ, the above embedding of distributions becomes injective [52]. Therefore, any two distinct distributions $p(X)$ and $q(X)$ are mapped to two distinct points in the feature space. We can treat the injective embedding μ_X as a sufficient statistic of the corresponding probability density. In other words, μ_X preserves all the information of $p(X)$. Using μ_X, we can uniquely recover $p(X)$ and any mathematical operation on $p(X)$ will have an equivalent operation on μ_X. These properties lead to the following equivalence relations. We can compute a functional $f : \mathcal{P} \rightarrow \mathbb{R}$ of the density $p(X)$ using only its embedding,

$$f(p(x)) = \tilde{f}(\mu_X) \tag{2.17}$$

by defining $\tilde{f} : \mathcal{F} \rightarrow \mathbb{R}$ as the operation on μ_X equivalent to f. Similarly, we can generalize this property to operators. An operator $\mathcal{T} : \mathcal{P} \rightarrow \mathbb{R}^{\mathrm{d}}$ applied to a density can also be equivalently carried out using its embedding,

$$\mathcal{T} \circ p(x) = \tilde{\mathcal{T}} \circ \mu_X \tag{2.18}$$

where $\tilde{\mathcal{T}} : \mathcal{F} \rightarrow \mathbb{R}^{\mathrm{d}}$ is again the corresponding equivalent operator applied to the embedding. In our derivations, we assume that there exists a feature space where the embeddings are injective and apply the above equivalence relations in subsequent sections.

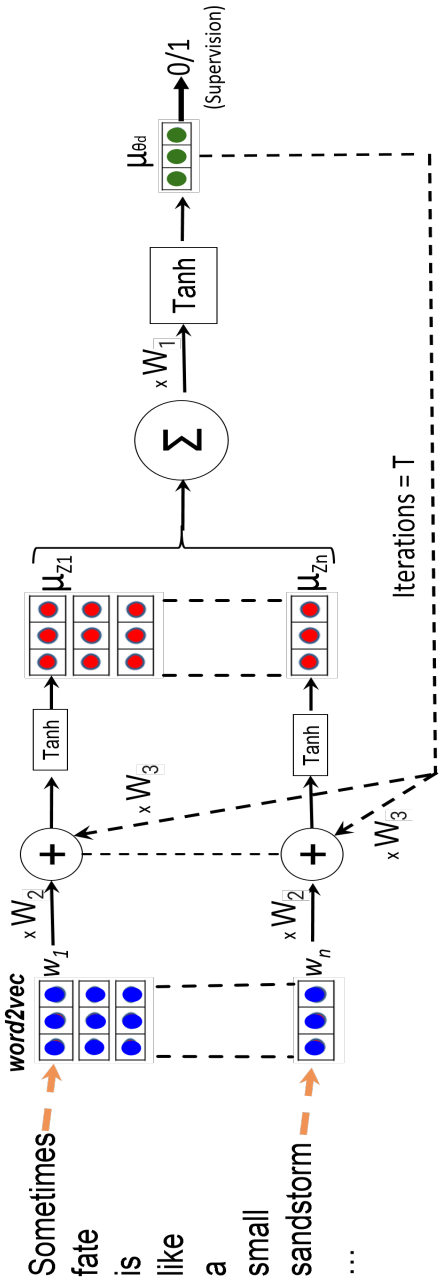

Figure 2.3: Visualization of the CoNN-sLDA architecture for a single document. For the i'th word, the latent topic variable is z_i. The embedding for the distribution $p(z_i)$ is μ_{z_i}; these embeddings are shown as three-dimensional vectors for illustration. They are accumulated and passed through a non-linearity to obtain μ_θ, which is the embedding of $p(\theta)$, the distribution over the topics for the document. Thus, the embedding μ_θ is determined (up to the non-linearity) by the average of the embeddings μ_{z_i}, as in the original LDA model. Similarly, there is feedback from μ_θ (which happens for T iterations, see Alg1), so that μ_θ, in turn, influences μ_{z_i}, again, as in the original LDA model.

2.3.5 Hilbert space embedding for LDA

We consider Hilbert space embeddings of $q(\theta)$, $q_i(z_i)$, as well as the equations (2.14) and (2.15). By definition given in equation(2.16),

$$\mu_\theta = \int_\theta \phi(\theta) q(\theta) d\theta \qquad \mu_{z_i} = \int_{z_i} \phi(z_i) q_i(z_i) dz_i \tag{2.19}$$

The variational update equations in (2.14) and (2.15) provide us with the key relationships between latent variables in the model. We can replace the specific distributional forms in these equations with operators that maintain the same relationships among distributions represented in the Hilbert space embeddings.

$$q(\theta) = f_1(\theta, \{q_i(z_i)\}) \quad q_i(z_i) = f_2(z_i, w_i, q(\theta)) \tag{2.20}$$

Here, f_1 and f_2 represent the abstract structure of the model implied by (2.14) and (2.15) without specific distributional forms. We will provide a specific instantiation of f_1 and f_2 shortly. Following the same argument as in equation (2.17), we can write equation (2.20) as $q(\theta) = \tilde{f}_1(\theta, \{\mu_{z_i}\})$. Similarly, $q_i(z_i) = \tilde{f}_2(z_i, w_i, \mu_\theta)$. Iterating through all values of θ, z_i and using the operator view given in equation (2.18) as reference, we get the following equivalent fixed-point equations in the Hilbert Space:

$$\mu_\theta = \mathcal{T}_1 \circ \{\mu_{z_i}\} \qquad \mu_{z_i} = \mathcal{T}_2 \circ [w_i, \mu_\theta] \tag{2.21}$$

2.3.6 Parameterization of Hilbert space embedding using Deep Neural Networks

The operators $\mathcal{T}_1$ and $\mathcal{T}_2$ have complex non-linear dependencies on the unknown true probability distributions and the feature map ϕ. Thus, we need to model these operators in such a way that we can utilize the available data to learn the underlying non-linear functions. We will use deep neural networks which are known for their ability to model non-linear

functions.

We start by parameterizing the embeddings. We assume that any point in the Hilbert space is a vector $\mu_i \in \mathbb{R}^{\mathrm{D}}$. Next, as the operators are non-linear function maps, we replace them by deep neural networks. In its simplest form, we only use a single fully connected layer with '$\tanh$' activations yielding the following fixed point update equations,

$$\mu_\theta = \tanh(\ W_1 \cdot \sum_{i=1}^{N} \{\mu_{z_i}\}\ ) \tag{2.22}$$

$$\mu_{z_i} = \tanh(\ W_2 \cdot word2vec(w_i) + W_3.\mu_\theta\ ) \tag{2.23}$$

The original work on Hilbert space embeddings required the embeddings to be injective. We observe that we do not need the embedding to be injective on the domain of all distributions. Instead, we only need it to be injective on the sub-domain of distributions used in the training corpus. The supervised training process on the training set will have to find embeddings that allow the model to distinguish documents that occur in the corpus automatically causing the learned embeddings to be injective for the training domain.

We keep the dimension of the $word2vec$ [53] embedding identical to the Hilbert space embedding, i.e. $w_i \in \mathbb{R}^{\mathrm{D}}$. Note, that the above parameterization is one example. Multiple fully connected layers can be used to achieve denser models.

Assume the parameters $word2vec$, W_1, W_2 and W_3 are known. We calculate the set of embeddings for a given text corpus by iterating equations(2.22, 2.23). Algorithm 1 summarizes this procedure. We normalize the embeddings after every iteration to avoid overflow. This is the heart of the Cooperative Neural Network paradigm in which a set of neural networks co-constrain each other to produce an embedding informed by prior structure. In our experience, we found that '$\tanh$' works better than 'σ' as a choice for non-linearity. Using rectified linear 'ReLU' units will not work as they zero out negative values of the embeddings. We apply dropout [46] to μ_{z_i}'s, μ_θ and *word2vec* for regularization.

Algorithm 1: Getting Hilbert Space Embeddings

Input: Parameters $\{W_1, W_2, W_3\}$
Initialize $\{\mu_\theta^{(0)}, \mu_{z_i}^{(0)}\} = \mathbf{0} \in \mathbb{R}^{\mathrm{D}}$.
for $t = 1$ **to** T iterations **do**
  for $i = 1$ **to** N words **do**
    $\mu_{z_i}^{(t)} = \tanh(W_2.word2vec(w_i) + W_3.\mu_\theta^{(t)})$
    Normalize $\mu_{z_i}^{(t)}$
  end for
  $\mu_\theta^{(t)} = \tanh(W_1.\sum_{i=1}^{N}\{\mu_{z_i}^{(t-1)}\})$
  Normalize $\mu_\theta^{(t)}$
end for
return $\{\mu_\theta^{(T)}\}$: Document embeddings

Algorithm 2: Training using Hilbert Space Embeddings

Input: Document Corpus $\mathcal{D}$, with each doc 'd' has set of words $[w_{d,i}] \in N_d$.
Initialize $\mathbf{P^{(0)}} = \{\mathbf{W^{(0)}}, \mathbf{u^{(0)}}, \mathbf{word2vec^{(0)}}\}$ with random values. Let 'learning rate = r'.
for $t = 1$ **to** $\mathcal{T}$ **do**
  Sample docs from $\mathcal{D}$ as $\{D_s, y_s\}$
  Using Alg(1) get Hilbert embeddings $\{\mu_{\theta_d}^s\}$ for 'D_s'
  $y_{pred} = \mathcal{H}\left(\mu_{\theta_d}^s; \mathbf{P^{(t-1)}}\right)$
  Update: $\mathbf{P^{(t)}}$ = $\mathbf{P^{(t-1)}}$ - r. $\bigtriangledown_{\mathbf{P^{(t-1)}}} L(y_{pred}, y_s)$
end for
return $\{\mathbf{P}^{\mathcal{T}}\}$

For every document, the algorithm returns the associated μ_θ embedding, representing the document in the Hilbert space.

In practice, the parameters $word2vec$, W_1, W_2 and W_3 are not known and need to be learned from training data. This requires formulating an objective function, and then optimizing that objective function. An additional advantage of the proposed method is that it allows using a wide variety of objective functions. In our case, we trained the model using a discriminative/supervised criterion that relies on the labels associated with each document, and we used binary cross-entropy loss or cross-entropy loss for multiclass classification.

Algorithm 2 summarizes the training procedure. It uses Algorithm 1 as a subroutine. The $\mathcal{H}$ function is chosen to be a single fully connected layer in our implementation,

which transforms the input embedding to a vector corresponding to number of classes. We sample (without replacement) a batch of documents D_s from the corpus, compute their embeddings and update the parameters. The loss function takes in the μ_θ embeddings and the corresponding document labels. The resulting model, called 'CoNN-sLDA' is schematically illustrated in Fig. 2.3.

The CoNN-sLDA model retains the overall structure of the LDA model by separating the problem into document topic distributions and word topic distributions within each document. As with traditional LDA, one can visualize a document corpus by projecting topic vectors associated with documents into a 2D plane (e.g., using MDS, tSNE). An advantage of CoNN-sLDA over typical neural network approaches is that typical DNNs produce only a single embedding, whereas CoNN-sLDA elegantly factors the local and global information into separate parts of the model. An advantage of CoNN-sLDA over traditional probabilistic graphical models is that we can use low-bias, highly expressive distributions implied by the neural network implementations of update operators.

2.3.7 Interpretability: Getting the relevant words based on embeddings obtained

The embedding model defines a relationship between words found in documents w and topic distributions for documents μ_θ. Usually we calculate the topic distribution from the words in documents. For interpretation purposes, we might wish to go the other direction: from topics μ_θ to words in the topic. For instance, after running CoNN-sLDA, we get embeddings for all of the documents. We could then cluster these to get K clusters. We might then ask how to interpret these clusters. We could take the mean embedding of each cluster μ_k and recover the words that would be associated with the cluster (e.g., SLR, aperture, resolution versus click-and-shoot, special effects). Alternatively, we could run PCA on the embedding space to find the principle directions of variation of document topics. We can then recover the words associated with the end-points of each distribution in order to label this dimension (e.g., light weight versus heavy or easy-to-use versus complicated).

We show here how to define a relationship between a given μ_θ and the words associated with the topic. Given the μ_θ from CoNN-sLDA model of a document under consideration, we want to find the top $word2vec$ vectors which satisfies the equation(2.24). If we substitute 2.23 into 2.22, we can eliminate the dependence on word topic distributions z_i.

$$\mu_\theta = \tanh(\ W_1 \cdot \sum_{i=1}^{N} \{\tanh(\ W_2 \cdot word2vec(w_i) + W_3.\mu_\theta\ )\}\ ) \tag{2.24}$$

The μ_θ terms are related by the sum of the embeddings of words in the text. The embeddings for the same word are always the same, so we can group all embeddings for word class c together and just keep a class weights F_c. We set Eq. 2.25 to zero so that we have an equation that measures discrepancy between current system and a consistent system. We then form an objective J_w which is a function of topic distribution μ_θ and K word class weights F_c.

$$J_w(F_c; \mu_\theta) = \tanh(\ W_1 \cdot \sum_{c=1}^{K} F_c\{\tanh(\ W_2 \cdot word2vec(w_c) + W_3.\mu_\theta\ )\}\ ) - \mu_\theta \tag{2.25}$$

Minimizing the square of J_w w.r.t. the F_c parameter will give us the weights of the words relevant to the embeddings μ_θ.

$$F_c^* = argmin_{F_c} J_w^2(F_c; \mu_\theta) \tag{2.26}$$

We can thus find the top most commonly occurring highly weighted words corresponding to any documents distribution embedding μ_θ or examine words associated with any μ_θ in the embedded space.

2.4 Experiments

2.4.1 Description of Datasets

We evaluated our model 'CoNN-sLDA' on two real-world datasets. The first dataset is a multi-domain sentiment dataset (MultiSent) [54], consisting of 342,104 Amazon product reviews on 25 different types of products (apparels, books, DVDs, kitchen appliances, $\cdots$). For each review, we go through the ratings given by the customer (between 1 to 5 stars) and label a it as positive, if the rating is higher than 3 stars and negative otherwise. We pose this as a binary classification problem. The average length of reviews is roughly 210 words after preprocessing the data. The ratio of positive to negative reviews is $\sim 8:1$. We use 5-fold cross validation and report the average area under the ROC curve (AUC), in $\%$.

The second dataset is the 20 Newsgroup dataset[1]. It has around 19,000 news articles, divided roughly equally into 20 different categories. We pose this as a multiclass classification problem and report accuracy over 20 classes. The dataset is divided into training set (11,314 articles) and test set (7,531 articles), approximately maintaining the relative ratio of articles of different categories. The average length of documents after preprocessing is ~ 160 words. This task becomes challenging as there are some categories that are highly similar, making their separation difficult. For example, the categories "PC hardware" and "Mac hardware" have quite a lot in common.

We apply standard text preprocessing steps to both datasets. We convert everything to lower case characters and remove the standard stopwords defined in the 'Natural Language Toolkit' library. We remove punctuations, followed by lemmatization and stemming to further clean the data. However, for other classifiers, we use the preprocessing techniques recommended by the respective authors.

[1] http://qwone.com/ jason/20Newsgroups/

2.4.2 Baselines for comparison

We compare 'CoNN-sLDA' with existing state-of-the-art algorithms for document classification. We compare against VI-sLDA, [28, 27], which includes the label of the document in the graphical model formulation and then maximizes the variational lower bound. Different from VI-sLDA, the supervised topic model using DiscLDA [29] reduces the dimensionality of topic vectors θ for classification by introducing a class-dependent linear transformation.

Boltzmann Machines are traditionally used to model distributions and with the recent development of deep learning techniques, these approaches have gained momentum. We compare with one such Deep Boltzmann Machine developed for modeling documents called Over-Replicated Softmax (OverRep-S) [32]. Another popular approach is by [44], called BP-sLDA, which does end-to-end learning of LDA by mirror-descent back propagation over a deep architecture. We also compare with a recent deep learning model developed by [45] called DUI-sLDA.

2.4.3 Classification Results

Table(2.1) shows the accuracy results on newsgroup dataset together with standard error on the mean (SEM) over 5 folds. For each of 5 folds, we split training data into train and validation and optimize all parameters. We then evaluate against a fixed common test set. As the number of classes is 20, we found that using higher Hilbert space dimensions work better (See entries for Dim=40 and Dim=80 in table). A dropout of ~ 0.8 was applied to *word2vec* embeddings. The batch size was fixed at 100 and we trained for around 400 batches. The performance of CoNN-sLDA is better than BP-sLDA and at par with 5 layer DUI-sLDA model. The cost sensitive version CoNN-sLDA (Imb), balances out the misclassification cost for different classes in the loss function tends to perform slightly better. The 20 newsgroup dataset is one of the earliest and most studied text corpuses. It is fairly separable, so most modern state-of-the-art methods do well on it, but it is an important benchmark to establish the credibility of an algorithm.

Table 2.1: '20 Newsgroups' classification accuracy on 19K documents. SEM over 5 fold CV. Dim indicates Hilbert space dimension.

Classifier	Accuracy(%)	Details
VI-sLDA	73.8± 0.49	K=50
DiscLDA	80.2± 0.45	K=50
OverRep-S	69.5± 0.36	K=512
BP-sLDA	81.8± 0.36	K=50, L=5
DUI-sLDA	83.5± 0.22	K=50, L=5
CoNN-sLDA	83.4 ± 0.18	Dim=40
CoNN-sLDA(imb)	**83.7± 0.13**	Dim=80

Table 2.2: 'MultiSent' AUC on 324K documents. SEM over 5 Fold CV. Dim indicates Hilbert space dimension.

Classifier	AUC (%)	Details
VI-sLDA	76.8± 0.40	K=50 (topics)
DiscLDA	82.1± 0.40	K=50
BP-sLDA	88.9± 0.36	K=50, L=5
DUI-sLDA	86.0± 0.31	K=50, L=1
DUI-sLDA	91.4± 0.27	K=50, L=5
CoNN-sLDA	**93.3± 0.13**	Dim=10
CoNN-sLDA(imb)	**93.4± 0.13**	Dim=20

Our CoNN-sLDA model was able to outperform the recently proposed state-of-the-art method, DUI-sLDA, on the large 'MultiSent' dataset (table2.2) having over 300K documents by a significant AUC margin of 2%. This corresponds to a 23% reduction in error rate. We used a single fully connected layer with $\tanh$ non-linear function for both, μ_θ, μ_{z_i} embeddings. Hilbert space dimension and *word2vec* dimension are both 10. We use a dropout probability of 0.1, The Algorithm(1) was unrolled for 1 iteration. 'Batch size' was set at 100 and ran for 3000 batches with optimization done using 'Adam' optimizer. We also ran a cost sensitive version of CoNN-sLDA (Imb) model, with a balancing ratio of 1.4 towards the minority class which was incorporated in the loss function. We observe slight improvement in results. CoNN-sLDA consistently outperformed other models over various choices of model parameters, see Section 2.4.4.

The number of layers required by other deep models like DUI-sLDA, BP-sLDA for good

classification is usually quite high and their performance decreases considerably with fewer layers. CoNN-sLDA outperforms them with a single layer neural network.

We have a vectorized and efficient implementation of CoNN-sLDA in PyTorch and Tensorflow. The results shown above are from the PyTorch version. We ran our experiments on NVIDIA Tesla P100 GPUs. The runtime for 1 fold of 'MultiSent' for the settings mentioned above is around *5 minutes*, while a single fold for '20 Newsgroup' dataset runs within *2 minutes*.

In Section 2.4.4, we report our experiments to optimize the algorithmic and architectural hyperparameters. We use the 'MultiSent' data for our analysis. In general for training, we recommend starting with a small Hilbert space dimension and batch size, then try increasing the number of fully connected layers and finally choose to unroll the model further.

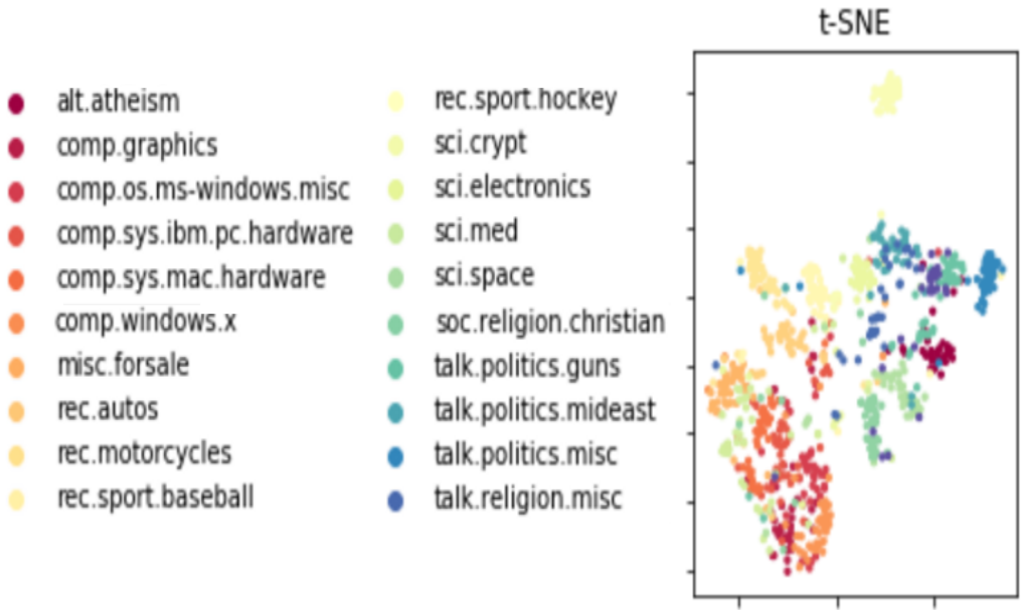

Figure 2.4: A t-SNE projection of the 40-dimensional embeddings μ_θ for test documents in the 20-Newsgroups dataset. The colors represent the category label for each document. The embeddings separate categories very well.

2.4.4 Architecture choices of CoNN-sLDA model

In this section we report on our experiments to optimize the algorithmic and architectural hyperparameters. We use the 'MultiSent' dataset for our analysis.

Varying Hilbert Space Embeddings dimension

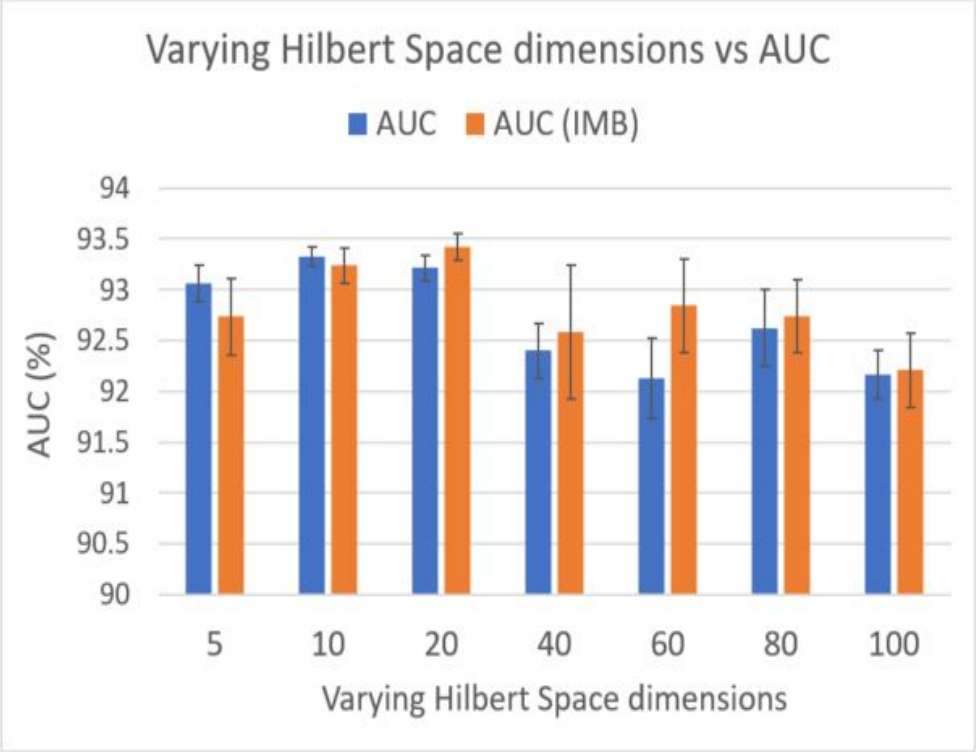

Figure 2.5: Varying Hilbert space embeddings dimension along the x-axis and the AUC values on y-axis. We also compare with the cost sensitive learning version, denoted by AUC (IMB) for every dimension choice. The depth of neural networks for both the embeddings μ_θ and μ_{z_i} is a single fully connected layer.

The dimensionality of the Hilbert space trades off the expressive power against computation and storage requirements of the model. In Fig. 2.5, we show varying Hilbert space dimensions on the x-axis and compare their AUCs. We observe a decline in AUC after Hilbert dimension of 20. We postulate that higher Hilbert space dimensions tend to overfit the data. Empirically we found that with lower Hilbert space dimensions we have to scale down the dropout appropriately.

As the data is imbalanced between number of positive and negative reviews, we did cost sensitive learning in CoNN-sLDA (Imb) by adjusting the weights of the loss function for different classes and were able to attain slight improvement. There can be a number of other techniques that can be used to handle such imbalanced datasets and that are compatible with Cooperative neural networks framework. For instance, sythetic minority over-sampling technique (SMOTE) [55] can be used as a preprocessing step. In this technique, the points belonging to the minortiy class (class having significantly less number of data points) are increased by sampling similar points around the minority class. This is shown to be a very effective technique and there are lots of variants of SMOTE which have been developed for datasets occurring in different domains, different data types like categorical, numerical or real data as well as the input dimensions of the data. Another alternative approach is to try undersampling the majority data points around the minority class. Though, these set of techniques are less popular and empirically are not as successful as over-sampling based approaches.

Varying number of Iterations of update equations in Algorithm(1)

Fig. 2.6 shows the plot of varying number of iterations of update equations in algorithm(1) versus the AUC obtained. We can observe the that AUC decreases and the corresponding standard deviation increases as we increase the number of iterations. In our experience, our algorithm works well even for a single iteration and going beyond 5 iteration gives no significant improvement in results.

Varying depth of the model

In Algorithm 1, we parameterized the embeddings μ_θ and μ_{z_i} using deep neural networks. Here, we analyze the results of varying the depth of the neural networks and their effect on the corresponding AUC. Fig. 2.7 shows a combination plot, where we visualize the AUC values for various different combinations of depth between μ_θ and μ_{z_i}.

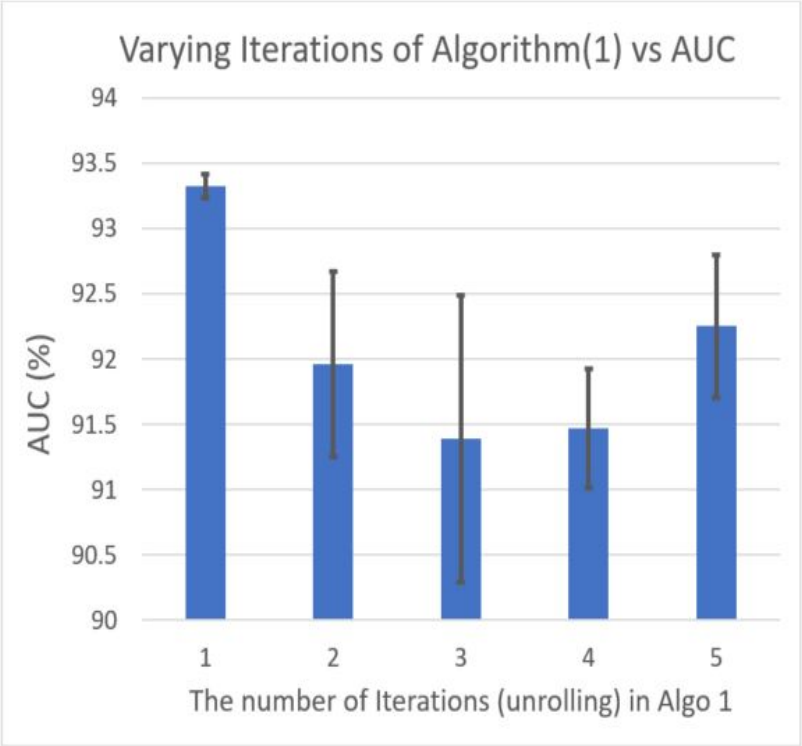

Figure 2.6: Varying iterations: Unrolling the model along the x-axis and AUC on the y-axis. For the 'MultiSent' dataset, we found that even using a single iteration works well.

We found that two fully connected layers for embedding μ_{z_i} and a single fully connected layer for embedding μ_θ works well for both datasets. Deeper models tend to overfit the data. For training, we recommend starting with a small Hilbert space dimension and batch size, then increase the number of fully connected layers, and finally choose to unroll the model further.

2.5 Discussions

In addition to supervised classification, we can use LDA style models for visualizing and interpreting the cluster structure of the datasets. For example, in CoNN-sLDA model, we can use t-SNE [56] to visualize the documents using their μ_θ values. In Fig. 2.4 we see that CoNN-sLDA clearly maps different newsgroups to homogeneous regions of space that help classification accuracy and provide insight into the structure of the domain. Similarly, Fig. 2.8 shows that CoNN-sLDA maps the positive and negative product reviews into

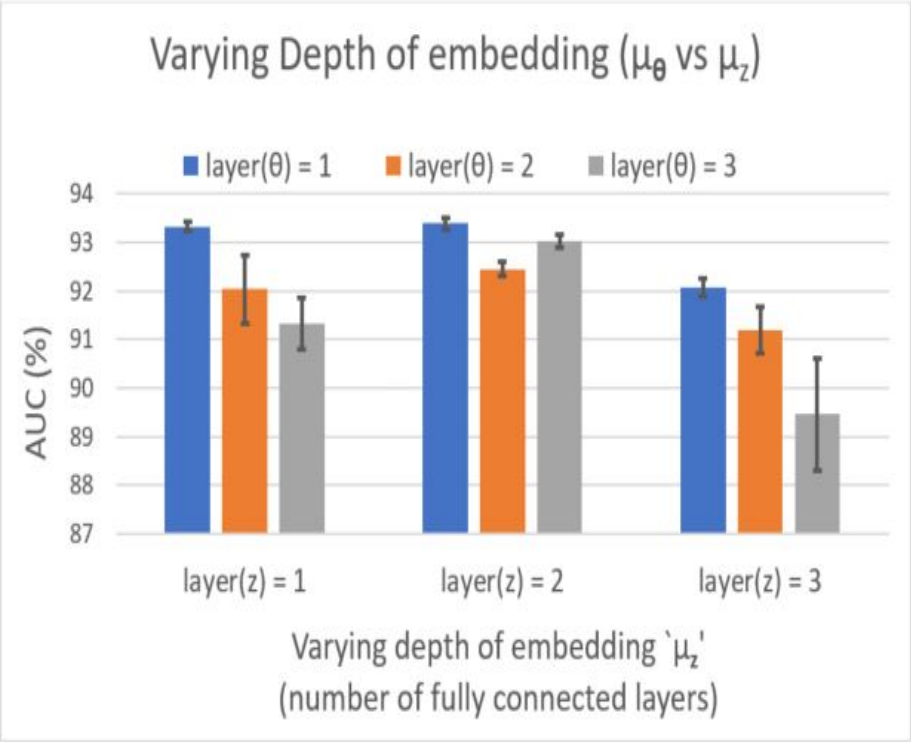

Figure 2.7: Varying μ_θ and μ_{z_i} : Plot showing number of fully connected layers for various combinations of μ_θ and μ_{z_i}. The AUC values are shown on y-axis. We observe that the setting where there are two fully connected layer of embedding μ_{z_i} consistently gives good results for varying layers of embedding μ_θ.

different regions facilitating classification and interpretation.

An interesting extension for the CoNN-sLDA model will be to map the Hilbert space topic embedding μ_θ back to the original topic space distribution. This would potentially allow us to provide text labels for the discovered clusters providing an intuitive interpretation for the model learned by our technique. Section 2.3.7 discusses an approach to get most relevant words in a document pertaining to a discriminative task.

Furthermore, the Cooperative neural network modules can be easily plugged in existing neural network based pipelines as it is a differentiable model which supports backpropagation of gradients. This makes Cooperative neural networks attractive as they can help in improving existing architectures and integrate into already existing machine learning systems.

In this work, we obtain the fixed point update equations using the mean-field inference

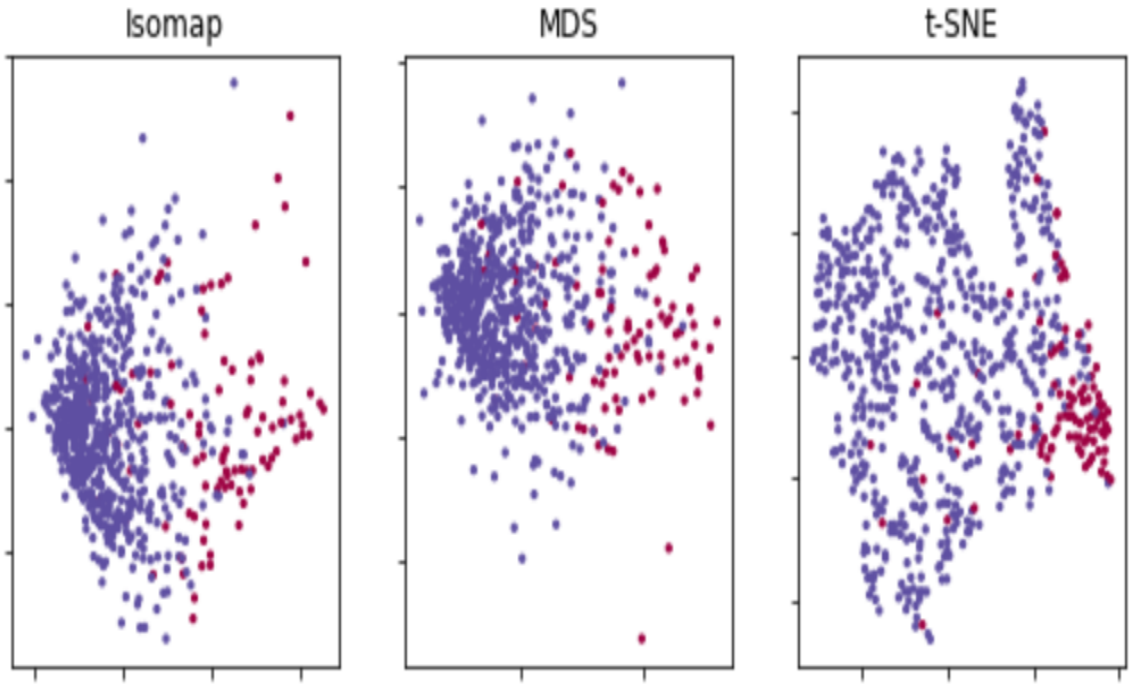

Figure 2.8: ISOMAP, MDS & tSNE visualization of a random sample of 10-dimensional μ_θ embeddings for Multisent documents (Blue positive, red negative). The embeddings project distinct categories to highly coherent regions.

technique. In general, we can extend this procedure to other variational inference techniques. For example, we can find embeddings for Algorithm 1 by minimizing the free energies of loopy belief propagation or its variants (e.g., [57]) and use Algorithm 2 to train them end-to-end.

2.6 Conclusion

Cooperative neural networks (CoNN) are a new theoretical approach for implementing learning systems which can exploit both prior insights about the independence structure of the problem domain and the universal approximation capability of deep networks. We make the theory concrete with an example, CoNN-sLDA, which has superior performance to both prior work based on the probabilistic graphical model LDA and generic deep networks. While we demonstrated the method on text classification using the structure of LDA, the approach provides a fully general methodology for computing factored embeddings using a

set of highly expressive networks. Furthermore, the CoNN models are fully differentiable modules and can be integrated into existing machine learning systems. Cooperative neural networks thus expand the design space of deep learning machines in new and promising ways.

CHAPTER 3

GLAD: LEARNING SPARSE GRAPH RECOVERY

3.1 Introduction

Recovering sparse conditional independence graphs from data is a fundamental problem in high dimensional statistics and time series analysis, and it has found applications in diverse areas. In computational biology, a sparse graph structure between gene expression data may be used to understand gene regulatory networks; in finance, a sparse graph structure between financial time-series may be used to understand the relationship between different financial assets. A popular formulation of the problem is an ℓ_1 regularization log-determinant estimation of the precision matrix. Based on this convex formulation, many algorithms have been designed to solve this problem efficiently, and one can formally prove that under a list of conditions, the solution of the optimization problem is guaranteed to recover the graph structure with high probability.

However, convex optimization based approaches have their own limitations. The hyper-parameters, such as the regularization parameters and learning rate, may depend on unknown constants, and need to be tuned carefully to achieve the recovery results. Furthermore, the formulation uses a single regularization parameter for all entries in the precision matrix, which may not be optimal. It is intuitive that one may obtain better recovery results by allowing the regularization parameters to vary across the entries in the precision matrix. However, such flexibility will lead to a quadratic increase in the number of hyperparameters, but it is hard for traditional approaches to search over a large number of hyperparameters. Thus, a new paradigm may be needed for designing more effective sparse recovery algorithms.

Recently, there has been a surge of interest in a new paradigm of algorithm design, where algorithms are augmented with learning modules trained directly with data, rather than prescribing every step of the algorithms. This is meaningful because very often a

family of optimization problems needs to be solved again and again, similar in structures but different in data. A data-driven algorithm may be able to leverage this distribution of problem instances, and learn an algorithm which performs better than traditional convex formulation. In our case, the sparse graph recovery problem may also need to be solved again and again, where the underlying graphs are different but have similar degree distribution, the magnitude of the precision matrix entries, etc. For instance, gene regulatory networks may be rewiring depending on the time and conditions, and we want to estimate them from gene expression data. Company relations may evolve over time, and we want to estimate their graph from stock data. Thus, we will also explore data-driven algorithm design in this paper.

Given a task (e.g. an optimization problem), an algorithm will solve it and provide a solution. Thus we can view an algorithm as a function mapping, where the input is the task-specific information (i.e. the sample covariance matrix in our case) and the output is the solution (i.e. the estimated precision matrix in our case). However, it is very challenging to design a data-driven algorithm for precision matrix estimation. First, the input and output of the problem may be large. A neural network parameterization of direct mapping from the input covariance matrix to the output precision matrix may require as many parameters as the square of the number of dimensions. Second, there are many structure constraints in the output. The resulting precision matrix needs to be positive definite and sparse, which is not easy to enforce by a simple deep learning architecture. Third, direct mapping may result in a model with lots of parameters, and hence may require lots of data to learn. Thus a data-driven algorithm needs to be designed carefully to achieve a better bias-variance trade-off and satisfy the output constraints.

In this paper, we propose a deep learning model '`GLAD`' with following attributes:

- Uses an unrolled Alternating Minimization (AM) algorithm as an inductive bias.
- The regularization and the square penalty terms are parameterized as entry-wise functions of intermediate solutions, allowing `GLAD` to learn to perform entry-wise regularization update.

- Furthermore, this data-driven algorithm is trained with a collection of problem instances in a supervised fashion, by directly comparing the algorithm outputs to the ground truth graphs.

In our experiments, we show that the AM architecture provides very good inductive bias, allowing the model to learn very effective sparse graph recovery algorithm with a small amount of training data. In all cases, the learned algorithm can recover sparse graph structures with much fewer data points from a new problem, and it also works well in recovering gene regulatory networks based on realistic gene expression data generators.

Related works. [58] considers CNN based architecture that directly maps empirical covariance matrices to estimated graph structures. Previous works have parameterized optimization algorithms as recurrent neural networks or policies in reinforcement learning. For instance, [59] considered directly parameterizing optimization algorithm as an RNN based framework for learning to learn. [60] approach the problem of automating algorithm design from reinforcement learning perspective and represent any particular optimization algorithm as a policy. [61] learn combinatorial optimzation over graph via deep Q-learning. These works did not consider the structures of our sparse graph recovery problem. Another interesting line of approach is to develop deep neural networks based on unfolding an iterative algorithm [20, 62, 21]. [21] developed ALISTA which is based on unrolling the Iterative Shrinkage Thresholding Algorithm (ISTA). [63] developed 'ADMM-Net', which is also developed for compressive sensing of MRI data. Though these seminal works were primarily developed for compressive sensing applications, they alluded to the general theme of using unrolled algorithms as inductive biases. We thus identify a suitable unrolled algorithm and leverage its inductive bias to solve the sparse graph recovery problem.

3.2 Sparse Graph Recovery Problem and Convex Formulation

Given m observations of a d-dimensional multivariate Gaussian random variable $X = [X_1, \ldots, X_d]^\top$, the sparse graph recovery problem aims to estimate its covariance matrix

Σ^* and precision matrix $\Theta^* = (\Sigma^*)^{-1}$. The ij-th component of Θ^* is zero if and only if X_i and X_j are conditionally independent given the other variables $\{X_k\}_{k\neq i,j}$. Therefore, it is popular to impose an ℓ_1 regularization for the estimation of Θ^* to increase its sparsity and lead to easily interpretable models. Following [22], the problem is formulated as the ℓ_1-regularized maximum likelihood estimation

$$\widehat{\Theta} = \arg\min_{\Theta\in\mathcal{S}^d_{++}} \quad -\log(\det\Theta) + \mathrm{tr}(\widehat{\Sigma}\Theta) + \rho\,\|\Theta\|_{1,\mathrm{off}}, \tag{3.1}$$

where $\widehat{\Sigma}$ is the empirical covariance matrix based on m samples, $\mathcal{S}^d_{++}$ is the space of $d\times d$ symmetric positive definite matrices (SPD), and $\|\Theta\|_{1,\mathrm{off}} = \sum_{i\neq j}|\Theta_{ij}|$ is the off-diagonal ℓ_1 regularizer with regularization parameter ρ. This estimator is sensible even for non-Gaussian X, since it is minimizing an ℓ_1-penalized log-determinant Bregman divergence [64]. The sparse precision matrix estimation problem in Eq. (3.1) is a convex optimization problem which can be solved by many algorithms. We give a few canonical and advanced examples which are compared in our experiments:

G-ISTA. G-ISTA is a proximal gradient method, and it updates the precision matrix iteratively

$$\Theta_{k+1} \leftarrow \eta_{\xi_k\rho}(\Theta_k - \xi_k(\widehat{\Sigma} - \Theta_k^{-1})), \quad \text{where } [\eta_\rho(X)]_{ij} := \mathrm{sign}(X_{ij})(|X_{ij}| - \rho)_+. \tag{3.2}$$

The step sizes ξ_k is determined by line search such that Θ_{k+1} is SPD matrix [65].

ADMM. Alternating direction methods of multiplier [18] transform the problem into an equivalent constrained form, decouple the log-determinant term and the ℓ_1 regularization term, and result in the following augmented Lagrangian form with a penalty parameter λ:

$$-\log(\det\Theta) + \mathrm{tr}(\widehat{\Sigma}\Theta) + \rho\,\|Z\|_1 + \langle\beta, \Theta - Z\rangle + \tfrac{1}{2}\lambda\|Z-\Theta\|_F^2. \tag{3.3}$$

Taking $U := \beta/\lambda$ as the scaled dual variable, the update rules for the ADMM algorithm are

$$\Theta_{k+1} \leftarrow \big(-Y + \sqrt{Y^\top Y + (4/\lambda)I}\big)/2, \text{ where } Y = \widehat{\Sigma}/\lambda - Z_k + U_k \tag{3.4}$$

$$Z_{k+1} \leftarrow \eta_{\rho/\lambda}(\Theta_{k+1} + U_k), \quad U_{k+1} \leftarrow U_k + \Theta_{k+1} - Z_{k+1} \tag{3.5}$$

BCD. Block-coordinate decent methods [66] updates each column (and the corresponding row) of the precision matrix iteratively by solving a sequence of lasso problems. The algorithm is very efficient for large scale problems involving thousands of variables.

Apart from various algorithms, rigorous statistical analysis has also been provided for the optimal solution of the convex formulation in Eq. (3.1). [64] established consistency of the estimator $\widehat{\Theta}$ in Eq. (3.1) in terms of both Frobenius and spectral norms, at rate scaling roughly as $\|\widehat{\Theta} - \Theta^*\| = \mathcal{O}\big(((d+s)\log d/m)^{1/2} \big)$ with high probability, where s is the number of nonzero entries in Θ^*. This statistical analysis also reveal certain **limitations** of the convex formulation:

The established consistency is based on a set of **carefully chosen conditions**, including the lower bound of sample size, the sparsity level of Θ^*, the degree of the graph, the magnitude of the entries in the covariance matrix, and the strength of interaction between edge and non-edge in the precision matrix (or mutual incoherence on the Hessian $\Gamma^* := \Sigma^* \otimes \Sigma^*$) . In practice, it may be hard to a problem to satisfy these recovery conditions.

Therefore, it seems that there is still room for improving the above convex optimization algorithms for recovering the true graph structure. Prior to the data-driven paradigm for sparse recovery, since the target parameter Θ^* is unknown, the best precision matrix recovery method is to resort to a surrogate objective function (for instance, equation 3.1). Optimally tuning the unknown parameter ρ is a very challenging problem in practice. Instead, we can leverage the large amount of simulation or real data and design a learning algorithm that directly optimizes the loss in equation 3.18.

Furthermore, since the log-determinant estimator in Eq. (3.1) is NOT directly optimizing the recovery objective $\|\widehat{\Theta} - \Theta^*\|_F^2$, there is also a **mismatch** in the optimization objective and the final evaluation objective (refer to the first experiment in section 3.6.2). This increase the hope one may improve the results by directly optimizing the recovery objective with the algorithms learned from data.

3.3 Learning Data-Driven Algorithm for Graph Recovery

In the remainder of the paper, we will present a data-driven method to learn an algorithm for precision matrix estimation, and we call the resulting algorithm GLAD (stands for **G**raph recovery **L**earning **A**lgorithm using **D**ata-driven training). We ask the question of

> Given a family of precision matrices, is it possible to improve recovery results for sparse graphs by learning a data-driven algorithm?

More formally, suppose we are given n precision matrices $\{\Theta^{*(i)}\}_{i=1}^n$ from a family $\mathcal{G}$ of graphs and m samples $\{\boldsymbol{x}^{(i,j)}\}_{j=1}^m$ associated with each $\Theta^{*(i)}$. These samples can be used to form n sample covariance matrices $\{\widehat{\Sigma}^{(i)}\}_{i=1}^n$. We are interested in learning an algorithm for precision matrix estimation by solving a supervised learning problem, $\min_f \frac{1}{n}\sum_{i=1}^n \mathcal{L}(\texttt{GLAD}_f(\widehat{\Sigma}^{(i)}), \Theta^{*(i)})$, where f is a set of parameters in $\texttt{GLAD}(\cdot)$ and the output of $\texttt{GLAD}_f(\widehat{\Sigma}^{(i)})$ is expected to be a good estimation of $\Theta^{*(i)}$ in terms of an interested evaluation metric $\mathcal{L}$. The benefit is that it can directly optimize the final evaluation metric which is related to the desired structure or graph properties of a family of problems. However, it is a challenging task to design a good parameterization of $\texttt{GLAD}_f$ for this graph recovery problem. We will explain the challenges below and then present our solution.

3.3.1 Challenges in Designing Learning Models

In the literature on learning data-driven algorithms, most models are designed using traditional deep learning architectures, such as fully connected DNN or recurrent neural networks. But, for graph recovery problems, directly using these architectures does not work well due to the following reasons.

First, using a fully connected neural network is not practical. Since both the input and the output of graph recovery problems are matrices, the number of parameters scales at least quadratically in d. Such a large number of parameters will need many input-output training pairs to provide a decent estimation. Thus some structures need to be imposed in the network to reduce the size of parameters and sample complexity.

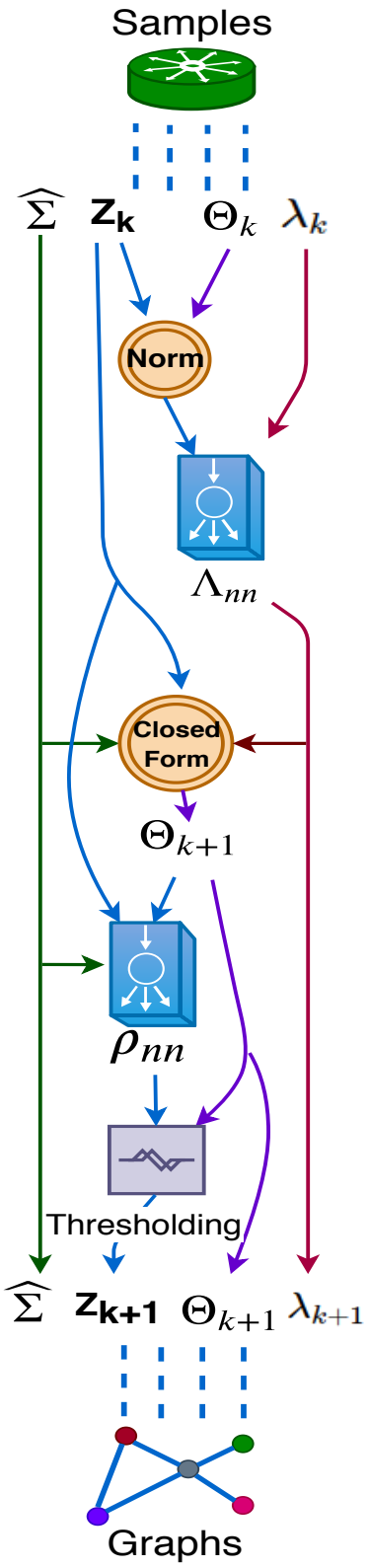

Figure 3.1: A recurrent unit `GLADcell`.

Second, structured models such as convolution neural networks (CNNs) have been applied to learn a mapping from $\widehat{\Sigma}$ to Θ^* [58]. Due to the structure of CNNs, the number of parameters can be much smaller than fully connected networks. However, a recovered graph should be permutation invariant with respect to the matrix rows/columns, and this constraint is very hard to be learned by CNNs, unless there are lots of samples. Also, the structure of CNN is a bias imposed on the model, and there is no guarantee why this structure may work.

Third, the intermediate results produced by both fully connected networks and CNNs are not interpretable, making it hard to diagnose the learned procedures and progressively output increasingly improved precision matrix estimators.

Fourth, the SPD constraint is hard to impose in traditional deep learning architectures.

Although, the above limitations do suggest a list of desiderata when designing learning models: Small model size; Minimalist learning; Interpretable architecture; Progressive improvement; and SPD output. These desiderata will motivate the design of our deep architecture using unrolled algorithms.

3.3.2 GLAD: Deep Learning Model based on Unrolled Algorithm

To take into account the above desiderata, we will use an unrolled algorithm as the template for the architecture design of GLAD. The unrolled algorithm already incorporates some problem structures, such as permutation invariance and interpretable intermediate results; but this unrolled algorithm does not traditionally have a learning component, and is typically not directly suitable for gradient-based approaches. We will leverage this inductive bias in our architecture design and augment the unrolled algorithm with suitable and flexible learning components, and then train these embedded models with stochastic gradient descent.

GLAD model is based on a reformulation of the original optimization problem in Eq. (3.1) with a squared penalty term, and an alternating minimization (AM) algorithm for it. More specifically, we consider a modified optimization with a quadratic penalty parameter λ:

$$\widehat{\Theta}_\lambda, \widehat{Z}_\lambda := \arg\min_{\Theta, Z \in \mathcal{S}^d_{++}} -\log(\det\Theta) + \mathrm{tr}(\widehat{\Sigma}\Theta) + \rho\,\|Z\|_1 + \tfrac{1}{2}\lambda\,\|Z-\Theta\|_F^2 \qquad (3.6)$$

and the alternating minimization (AM) method for solving it, the following subsection gives in-depth derivations for the AM method.

Derivation of Alternating Minimization Steps

Given the optimization problem

$$\widehat{\Theta}_\lambda, \widehat{Z}_\lambda := \arg\min_{\Theta, Z \in \mathcal{S}^d_{++}} -\log(\det\Theta) + \mathbf{tr}(\widehat{\Sigma}\Theta) + \rho\,\|Z\|_1 + \tfrac{1}{2}\lambda\,\|Z-\Theta\|_F^2\,, \quad (3.7)$$

Alternating Minimization is performing

$$\Theta^{\mathrm{AM}}_{k+1} \leftarrow \underset{\Theta \in \mathcal{S}^d_{++}}{\arg\min} -\log(\det\Theta) + \mathbf{tr}(\widehat{\Sigma}\Theta) + \tfrac{1}{2}\lambda\left\|Z^{\mathrm{AM}}_k - \Theta\right\|_F^2 \quad (3.8)$$

$$Z^{\mathrm{AM}}_{k+1} \leftarrow \underset{Z \in \mathcal{S}^d_{++}}{\arg\min}\, \mathbf{tr}(\widehat{\Sigma}\Theta^{\mathrm{AM}}_{k+1}) + \rho\,\|Z\|_1 + \tfrac{1}{2}\lambda\left\|Z - \Theta^{\mathrm{AM}}_{k+1}\right\|_F^2. \quad (3.9)$$

Taking the gradient of the objective function with respect to Θ to be zero, we have

$$-\Theta^{-1} + \widehat{\Sigma} + \lambda(\Theta - Z) = 0. \quad (3.10)$$

Taking the gradient of the objective function with respect to Z to be zero, we have

$$\rho\partial\ell_1(Z) + \lambda(Z - \Theta) = 0, \quad (3.11)$$

where

$$\partial\ell_1(Z_{ij}) = \begin{cases} 1 & Z_{ij} > 0, \\ -1 & Z_{ij} < 0, \\ [-1,1] & Z_{ij} = 0. \end{cases} \quad (3.12)$$

Solving the above two equations, we obtain:

$$\frac{1}{2}\big(-Y + \sqrt{Y^\top Y + \frac{4}{\lambda}I}\big) = \underset{\Theta \in \mathcal{S}^d_{++}}{\arg\min} -\log(\det\Theta) + \mathbf{tr}(\widehat{\Sigma}\Theta) + \frac{1}{2}\lambda\,\|Z-\Theta\|_F^2\,, \quad (3.13)$$

$$\text{where } Y = \frac{1}{\lambda}\widehat{\Sigma} - Z, \quad (3.14)$$

$$\eta_{\rho/\lambda}(\Theta) = \underset{Z \in \mathcal{S}^d_{++}}{\arg\min}\, \mathbf{tr}(\widehat{\Sigma}\Theta) + \rho\,\|Z\|_1 + \frac{1}{2}\lambda\,\|Z-\Theta\|_F^2\,. \quad (3.15)$$

Neural network based parameterization of AM updates

We end up with the following update equations after applying the AM method.

$$\Theta_{k+1}^{\text{AM}} \leftarrow \tfrac{1}{2}\big(-Y + \sqrt{Y^\top Y + \tfrac{4}{\lambda} I}\big), \text{ where } Y = \tfrac{1}{\lambda}\widehat{\Sigma} - Z_k^{\text{AM}}; \tag{3.16}$$

$$Z_{k+1}^{\text{AM}} \leftarrow \eta_{\rho/\lambda}(\Theta_{k+1}^{\text{AM}}), \tag{3.17}$$

where $\eta_{\rho/\lambda}(\theta) := \text{sign}(\theta)\max(|\theta| - \rho/\lambda, 0)$. The derivation of these steps are given in Section 3.3.2. We replace the penalty constants (ρ, λ) by problem dependent neural networks, ρ_{nn} and Λ_{nn}. These neural networks are minimalist in terms of the number of parameters as the input dimensions are mere $\{3, 2\}$ for $\{\rho_{nn}, \Lambda_{nn}\}$ and outputs a single value. Algorithm 3 summarizes the update equations for our unrolled AM based model, `GLAD`. Except for the parameters in ρ_{nn} and Λ_{nn}, the constant t for initialization is also a learnable scalar parameter. This unrolled algorithm with neural network augmentation can be viewed as a highly structured recurrent architecture as illustrated in Fig. 3.1.

Algorithm 3: GLAD

Function `GLADcell` $(\widehat{\Sigma}, \Theta, Z, \lambda)$ **:**

- $\lambda \leftarrow \Lambda_{nn}(\|Z - \Theta\|_F^2, \lambda)$
- $Y \leftarrow \lambda^{-1}\widehat{\Sigma} - Z$
- $\Theta \leftarrow \frac{1}{2}\big(-Y + \sqrt{Y^\top Y + \frac{4}{\lambda} I}\big)$
- **For** *all* i, j **do**
  - $\rho_{ij} = \rho_{nn}(\Theta_{ij}, \widehat{\Sigma}_{ij}, Z_{ij})$
  - $Z_{ij} \leftarrow \eta_{\rho_{ij}}(\Theta_{ij})$
- **return** Θ, Z, λ

Function `GLAD` $(\widehat{\Sigma})$ **:**

- $\Theta_0 \leftarrow (\widehat{\Sigma} + tI)^{-1}$, $\lambda_0 \leftarrow 1$
- **For** $k = 0$ *to* $K - 1$ **do**
  - $\Theta_{k+1}, Z_{k+1}, \lambda_{k+1}$
  - $\leftarrow$`GLADcell` $(\widehat{\Sigma}, \Theta_k, Z_k, \lambda_k)$
- **return** Θ_K, Z_K

There are many traditional algorithms for solving graph recovery problems. We choose AM as our basis because: First, empirically, we tried models built upon other algorithms including G-ISTA, ADMM, etc, but AM-based model gives consistently better performances. Sections 3.8.1 & 3.8.2 discuss different parameterizations tried. Second, and more importantly, the AM-based architecture has a nice property of maintaining Θ_{k+1} as a SPD matrix throughout the iterations as long as $\lambda_k < \infty$. Third, as we prove later in Section 3.4, the AM algorithm has linear convergence rate, allowing us to use a fixed small number of iterations and still achieve small error margins.

3.3.3 Training algorithm

To learn the parameters in `GLAD` architecture, we will directly optimize the recovery objective function rather than using log-determinant objective. A nice property of our deep learning architecture is that each iteration of our model will output a valid precision matrix estimation. This allows us to add auxiliary losses to regularize the intermediate results of our `GLAD` architecture, guiding it to learn parameters which can generate a smooth solution trajectory.

Specifically, we will use Frobenius norm in our experiments, and design an objective which has some resemblance to the discounted cumulative reward in reinforcement learning:

$$\min_{f} \quad \text{loss}_f := \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} \gamma^{K-k} \left\| \Theta_k^{(i)} - \Theta^* \right\|_F^2, \tag{3.18}$$

where $(\Theta_k^{(i)}, Z_k^{(i)}, \lambda_k^{(i)}) = \texttt{GLADcell}_f(\widehat{\Sigma}^{(i)}, \Theta_{k-1}^{(i)}, Z_{k-1}^{(i)}, \lambda_{k-1}^{(i)})$ is the output of the recurrent unit `GLADcell` at k-th iteration, K is number of unrolled iterations, and $\gamma \leq 1$ is a discounting factor.

We will use stochastic gradient descent algorithm to train the parameters f in the `GLADcell`. A key step in the gradient computation is to propagate gradient through the matrix square root in the `GLADcell`. To do this efficiently, we make use of the property of

SPD matrix that $X = X^{1/2}X^{1/2}$, and the product rule of derivatives to obtain

$$dX = d(X^{1/2})X^{1/2} + X^{1/2}d(X^{1/2}). \tag{3.19}$$

The above equation is a Sylvester's equation for $d(X^{1/2})$. Since the derivative dX for X is easy to obtain, then the derivative of $d(X^{1/2})$ can be obtained by solving the Sylvester's equation in (3.19).

The objective function in equation 3.18 should be understood in a similar way as in [20, 58, 21] where deep architectures are designed to directly produce the sparse outputs.

For GLAD architecture, a collection of input covariance matrix and ground truth sparse precision matrix pairs are available during training, either coming from simulated or real data. Thus the objective function in equation 3.18 is formed to directly compare the output of GLAD with the ground truth precision matrix. The goal is to train the deep architecture which can perform well for a family/distribution of input covariance matrix and ground truth sparse precision matrix pairs. The average in the objective function is over different input covariance and precision matrix pairs such that the learned architecture is able to perform well over a family of problem instances.

Furthermore, each layer of our deep architecture outputs an intermediate prediction of the sparse precision matrix. The objective function takes into account all these intermediate outputs, weights the loss according to the layer of the deep architecture, and tries to progressively bring these intermediate layer outputs closer and closer to the target ground truth.

3.3.4 A note on GLAD architecture's expressive ability

We note that the designed architecture, is more flexible than just learning the regularization parameters. The component in GLAD architecture corresponding to the regularization parameters are entry-wise and also adaptive to the input covariance matrix and the intermediate outputs. GLAD architecture can adaptively choose a matrix of regularization parameters. This task will be very challenging if the matrix of regularization parameters are tuned

manually using cross-validation. A recent theoretical work [67] also validates the choice of GLAD's design.

3.4 Theoretical Analysis

Since GLAD architecture is obtained by augmenting an unrolled optimization algorithm by learnable components, the question is what kind of guarantees can be provided for such learned algorithm, and whether learning can bring benefits to the recovery of the precision matrix. In this section, we will first analyze the statistical guarantee of running the AM algorithm in Eq. (3.16) and Eq. (3.17) for k steps with a fixed quadratic penalty parameter λ, and then interpret its implication for the learned algorithm. First, we need some standard assumptions about the true model from the literature [68]:

Assumption 1. *Let the set* $S = \{(i,j) : \Theta^*_{ij} \neq 0, i \neq j\}$. *Then card*$(S) \leq s$.

Assumption 2. $\Lambda_{\min}(\Sigma^*) \geq \epsilon_1 > 0$ *(or equivalently* $\Lambda_{\max}(\Theta^*) \leq 1/\epsilon_1$*),* $\Lambda_{\max}(\Sigma^*) \leq \epsilon_2$ *and an upper bound on* $\|\widehat{\Sigma}\|_2 \leq c_{\widehat{\Sigma}}$.

The assumption 2 guarantees that Θ^* exists. Assumption 1 just upper bounds the sparsity of Θ^* and does not stipulate anything in particular about s. These assumptions characterize the fundamental limitation of the sparse graph recovery problem, beyond which recovery is not possible. Under these assumptions, we prove the linear convergence of AM algorithm,

Theorem 1. *Under the assumptions 1 & 2, if* $\rho \asymp \sqrt{\frac{\log d}{m}}$*, where* ρ *is the* l_1 *penalty,* d *is the dimension of problem and* m *is the number of samples, the Alternate Minimization algorithm has linear convergence rate for optimization objective defined in (3.6). The* k^{th} *iteration of the AM algorithm satisfies,*

$$\left\|\Theta_k^{AM} - \Theta^*\right\|_F \leqslant C_\lambda \left\|\Theta_{k-1}^{AM} - \widehat{\Theta}_\lambda\right\|_F + \mathcal{O}_{\mathbb{P}}\left(\sqrt{\frac{(\log d)/m}{\min(\frac{1}{(d+s)}, \frac{\lambda}{d^2})}}\right), \tag{3.20}$$

where $0 < C_\lambda < 1$ *is a constant depending on* λ.

A complete proof is given in the following Section 3.4.1.

3.4.1 Linear Convergence Rate Analysis

Proof of Theorem

Theorem 1. *Under the assumptions 1 & 2, if* $\rho \asymp \sqrt{\frac{\log d}{m}}$*, where* ρ *is the* l_1 *penalty,* d *is the dimension of problem and* m *is the number of samples, the Alternate Minimization algorithm has linear convergence rate for optimization objective defined in (3.6). The* k^{th} *iteration of the AM algorithm satisfies,*

$$\left\|\Theta_k^{AM} - \Theta^*\right\|_F \leqslant C_\lambda \left\|\Theta_{k-1}^{AM} - \widehat{\Theta}_\lambda\right\|_F + \mathcal{O}_{\mathbb{P}}\left(\sqrt{\frac{(\log d)/m}{\min(\frac{1}{(d+s)}, \frac{\lambda}{d^2})}}\right), \tag{3.20}$$

where $0 < C_\lambda < 1$ *is a constant depending on* λ*.*

We will reuse the following notations for the sake of convenience and completeness:

$$\widehat{\Sigma}^m : \quad \text{sample covariance matrix based on } m \text{ samples}, \tag{3.21}$$

$$\mathcal{G}(\Theta; \rho) := -\log(\det\Theta) + \text{tr}(\hat{\Sigma}^m\Theta) + \rho\left\|\Theta\right\|_{1,\text{off}}, \tag{3.22}$$

$$\widehat{\Theta}_{\mathcal{G}} := \arg\min_{\Theta \in \mathcal{S}_{++}^d} \ \mathcal{G}(\Theta; \rho), \tag{3.23}$$

$$f(\Theta, Z; \rho, \lambda) := -\log(\det\Theta) + \text{tr}(\widehat{\Sigma}\Theta) + \rho\left\|Z\right\|_1 + \frac{1}{2}\lambda\left\|Z - \Theta\right\|_F^2, \tag{3.24}$$

$$\widehat{\Theta}_\lambda, \widehat{Z}_\lambda := \arg\min_{\theta, Z \in \mathcal{S}_{++}^d} f(\Theta, Z; \rho, \lambda), \tag{3.25}$$

$$f^*(\rho, \lambda) := \min_{\theta, Z \in \mathcal{S}_{++}^d} f(\Theta, Z; \rho, \lambda) = f(\widehat{\Theta}_\lambda, \widehat{Z}_\lambda; \rho, \lambda), \tag{3.26}$$

$$\eta_{\rho/\lambda}(\theta) := \text{sign}(\theta)\max(|\theta| - \rho/\lambda, 0). \tag{3.27}$$

The update rules for Alternating Minimization are:

$$\Theta_{k+1}^{\text{AM}} \leftarrow \tfrac{1}{2}\big(-Y + \sqrt{Y^\top Y + \tfrac{4}{\lambda}I}\big), \text{ where } Y = \tfrac{1}{\lambda}\widehat{\Sigma} - Z_k^{\text{AM}}; \tag{3.28}$$

$$Z_{k+1}^{\text{AM}} \leftarrow \eta_{\rho/\lambda}(\Theta_{k+1}^{\text{AM}}), \tag{3.29}$$

Assumptions: With reference to the theory developed in [68], we make the following assumptions about the true model. ($\mathcal{O}_{\mathbb{P}}(\cdot)$ is used to denote bounded in probability.)

Assumption 1. *Let the set* $S = \{(i, j) : \Theta_{ij}^* \neq 0, i \neq j\}$*. Then card*$(S) \leq s$*.*

Assumption 2. $\Lambda_{\min}(\Sigma^*) \geq \epsilon_1 > 0$ *(or equivalently* $\Lambda_{\max}(\Theta^*) \leq 1/\epsilon_1$*),* $\Lambda_{\max}(\Sigma^*) \leq \epsilon_2$ *and an upper bound on* $\|\widehat{\Sigma}\|_2 \leq c_{\widehat{\Sigma}}$.

We now proceed towards the proof:

Lemma 2. *For any* $x, y, k \in \mathbb{R}$*,* $k > 0$*,* $x \neq y$*,*

$$\frac{(\sqrt{x^2+k} - \sqrt{y^2+k})^2}{(x-y)^2} \leq 1 - \frac{1}{\sqrt{(\frac{x^2}{k}+1)(\frac{y^2}{k}+1)}}. \tag{3.30}$$

Proof.

$$\frac{(\sqrt{x^2+k} - \sqrt{y^2+k})^2}{(x-y)^2} = \frac{(x^2+k) + (y^2+k) - 2\sqrt{x^2+k}\sqrt{y^2+k}}{(x-y)^2} \tag{3.31}$$

$$= \frac{(x-y)^2 - 2(\sqrt{x^2+k}\sqrt{y^2+k} - (xy+k))}{(x-y)^2} \tag{3.32}$$

$$= 1 - 2\frac{(\sqrt{x^2+k}\sqrt{y^2+k} - (xy+k))(\sqrt{x^2+k}\sqrt{y^2+k} + (xy+k))}{(x-y)^2(\sqrt{x^2+k}\sqrt{y^2+k} + (xy+k))} \tag{3.33}$$

$$= 1 - 2\frac{k(x-y)^2}{(x-y)^2(\sqrt{x^2+k}\sqrt{y^2+k} + (xy+k))} \tag{3.34}$$

$$= 1 - \frac{2k}{\sqrt{x^2+k}\sqrt{y^2+k} + (xy+k)} \tag{3.35}$$

$$\leq 1 - \frac{1}{\sqrt{(\frac{x^2}{k}+1)(\frac{y^2}{k}+1)}} \tag{3.36}$$

□

Lemma 3. *For any* $X, Y \in \mathcal{S}^d$*,* $\lambda > 0$*,* $A(Y) = \sqrt{Y^\top Y + \frac{4}{\lambda}I}$ *satisfies the following inequality,*

$$\|A(X) - A(Y)\|_F \leq \alpha_\lambda \|X - Y\|_F, \tag{3.37}$$

where $0 < \alpha_\lambda = 1 - \frac{1}{2}(\frac{\lambda}{4}\Lambda_{max}(X)^2 + 1)^{-1/2}(\frac{\lambda}{4}\Lambda_{max}(Y)^2 + 1)^{-1/2} < 1$*,* $\Lambda_{max}(X)$ *is the largest eigenvalue of* X *in absolute value.*

Proof. First we factorize X using eigen decomposition, $X = Q_X^\top D_X Q_X$, where Q_X and

D_X are orthogonal matrix and diagonal matrix, respectively. Then we have,

$$A(X) = \sqrt{Q_X^\top D_X^2 Q_X + \frac{4}{\lambda} I} = \sqrt{Q_X^\top (D_X^2 + \frac{4}{\lambda} I) Q_X} = Q_X^\top \sqrt{D_X^2 + \frac{4}{\lambda} I} Q_X. \quad (3.38)$$

Similarly, the above equation holds for Y. Therefore,

$$\begin{aligned}
\|A(X) - A(Y)\|_F &= \left\| Q_X^\top \sqrt{D_X^2 + \frac{4}{\lambda} I} Q_X - Q_Y^\top \sqrt{D_Y^2 + \frac{4}{\lambda} I} Q_Y \right\|_F && (3.39)\\
&= \left\| Q_X (Q_X^\top \sqrt{D_X^2 + \frac{4}{\lambda} I} Q_X - Q_Y^\top \sqrt{D_Y^2 + \frac{4}{\lambda} I} Q_Y) Q_X^\top \right\|_F && (3.40)\\
&= \left\| \sqrt{D_X^2 + \frac{4}{\lambda} I} - Q_X Q_Y^\top \sqrt{D_Y^2 + \frac{4}{\lambda} I} Q_Y Q_X^\top \right\|_F && (3.41)\\
&= \left\| \sqrt{D_X^2 + \frac{4}{\lambda} I} - Q^\top \sqrt{D_Y^2 + \frac{4}{\lambda} I} Q \right\|_F, && (3.42)
\end{aligned}$$

where we define $Q := Q_Y Q_X^\top$. Similarly, we have,

$$\begin{aligned}
\|X - Y\|_F &= \left\| Q_X^\top D_X Q_X - Q_Y^\top D_Y Q_Y \right\|_F && (3.43)\\
&= \left\| D_X - Q_X Q_Y^\top D_Y Q_Y Q_X^\top \right\|_F && (3.44)\\
&= \left\| D_X - Q^\top D_Y Q \right\|_F. && (3.45)
\end{aligned}$$

Then the i-th entry on the diagonal of $Q^\top D_Y Q$ is $\sum_{j=1}^d D_{Yjj} Q_{ji}^2$. Using the fact that D_X and D_Y are diagonal, we have,

$$\begin{aligned}
\|X - Y\|_F^2 &= \left\| D_X - Q^\top D_Y Q \right\|_F^2 && (3.46)\\
&= \left\| Q^\top D_Y Q \right\|_F^2 - \sum_{i=1}^d (\sum_{j=1}^d D_{Yjj} Q_{ji}^2)^2 + \sum_{i=1}^d (D_{Xii} - \sum_{j=1}^d D_{Yjj} Q_{ji}^2)^2 && (3.47)\\
&= \|D_Y\|_F^2 + \sum_{i=1}^d D_{Xii} (D_{Xii} - 2 \sum_{j=1}^d D_{Yjj} Q_{ji}^2) && (3.48)\\
&= \sum_{i=1}^d (D_{Xii}^2 + D_{Yii}^2) - 2 \sum_{i=1}^d \sum_{j=1}^d D_{Xii} D_{Yjj} Q_{ji}^2 && (3.49)\\
&= \sum_{i=1}^d \sum_{j=1}^d Q_{ji}^2 (D_{Xii} - D_{Yjj})^2. && (3.50)
\end{aligned}$$

The last step makes use of $\sum_{i=1}^d Q_{ji}^2 = 1$ and $\sum_{j=1}^d Q_{ji}^2 = 1$. Similarly, using (3.42), we

have,

$$\|A(X) - A(Y)\|_F^2 = \left\| \sqrt{D_X^2 + \frac{4}{\lambda} I} - Q^\top \sqrt{D_Y^2 + \frac{4}{\lambda} I} Q \right\|_F^2 \tag{3.51}$$

$$= \sum_{i=1}^{d} \sum_{j=1}^{d} Q_{ji}^2 (\sqrt{D_{Xii}^2 + \frac{4}{\lambda}} - \sqrt{D_{Yjj}^2 + \frac{4}{\lambda}})^2 \tag{3.52}$$

Assuming $\|X - Y\|_F > 0$ (otherwise (3.37) trivially holds), using (3.52) and (3.50), we have,

$$\frac{\|A(X) - A(Y)\|_F^2}{\|X - Y\|_F^2} = \frac{\sum_{i=1}^{d} \sum_{j=1}^{d} Q_{ji}^2 (\sqrt{D_{Xii}^2 + \frac{4}{\lambda}} - \sqrt{D_{Yjj}^2 + \frac{4}{\lambda}})^2}{\sum_{i=1}^{d} \sum_{j=1}^{d} Q_{ji}^2 (D_{Xii} - D_{Yjj})^2} \tag{3.53}$$

$$\leq \max_{i,j=1,\cdots,d,\ D_{Xii} \neq D_{Yjj}} \frac{(\sqrt{D_{Xii}^2 + \frac{4}{\lambda}} - \sqrt{D_{Yjj}^2 + \frac{4}{\lambda}})^2}{(D_{Xii} - D_{Yjj})^2} \tag{3.54}$$

Using lemma (2), we have,

$$\frac{\|A(X) - A(Y)\|_F^2}{\|X - Y\|_F^2} \leq \max_{i,j=1,\cdots,d,\ D_{Xii} \neq D_{Yjj}} \frac{(\sqrt{D_{Xii}^2 + \frac{4}{\lambda}} - \sqrt{D_{Yjj}^2 + \frac{4}{\lambda}})^2}{(D_{Xii} - D_{Yjj})^2} \tag{3.55}$$

$$\leq \max_{i,j=1,\cdots,d,\ D_{Xii} \neq D_{Yjj}} 1 - \frac{1}{\sqrt{(\frac{D_{Xii}^2}{\frac{4}{\lambda}} + 1)(\frac{D_{Yjj}^2}{\frac{4}{\lambda}} + 1)}} \tag{3.56}$$

$$\leq 1 - \frac{1}{\sqrt{(\frac{\lambda}{4} \max_i D_{Xii}^2 + 1)(\frac{\lambda}{4} \max_j D_{Yjj}^2 + 1)}} \tag{3.57}$$

$$= 1 - (\frac{\lambda}{4} \Lambda_{max}(X)^2 + 1)^{-1/2} (\frac{\lambda}{4} \Lambda_{max}(Y)^2 + 1)^{-1/2}. \tag{3.58}$$

Therefore,

$$\frac{\|A(X) - A(Y)\|_F}{\|X - Y\|_F} \leq \sqrt{1 - (\frac{\lambda}{4} \Lambda_{max}(X)^2 + 1)^{-1/2} (\frac{\lambda}{4} \Lambda_{max}(Y)^2 + 1)^{-1/2}} \tag{3.59}$$

$$\leq 1 - \frac{1}{2} (\frac{\lambda}{4} \Lambda_{max}(X)^2 + 1)^{-1/2} (\frac{\lambda}{4} \Lambda_{max}(Y)^2 + 1)^{-1/2}. \tag{3.60}$$

□

Lemma 4. *Under assumption (2), the output of the k-th and $(k+1)$-th AM step Z_k^{AM}, Z_{k+1}^{AM} and Θ_{k+1}^{AM} satisfy the following inequality,*

$$\left\| Z_{k+1}^{AM} - \widehat{Z}_\lambda \right\|_F \leq \left\| \Theta_{k+1}^{AM} - \widehat{\Theta}_\lambda \right\|_F \leq C_\lambda \left\| Z_k^{AM} - \widehat{Z}_\lambda \right\|_F, \tag{3.61}$$

where $0 < C_\lambda < 1$ is a constant depending on λ.

Proof. The first part is easy to show, if we observe that in the second update step of AM (3.17), $\eta_{\rho/\lambda}$ is a contraction under metric $d(X,Y) = \|X - Y\|_F$. Therefore we have,

$$\left\| Z_{k+1}^{\text{AM}} - \widehat{Z}_\lambda \right\|_F = \left\| \eta_{\rho/\lambda}(\Theta_{k+1}^{\text{AM}}) - \eta_{\rho/\lambda}(\widehat{\Theta}_\lambda) \right\|_F \leq \left\| \Theta_{k+1}^{\text{AM}} - \widehat{\Theta}_\lambda \right\|_F . \tag{3.62}$$

Next we will prove the second part. To simplify notation, we let $A(X) = \sqrt{X^\top X + \frac{4}{\lambda} I}$. Using the first update step of AM (3.16), we have,

$$\left\| \Theta_{k+1}^{\text{AM}} - \widehat{\Theta}_\lambda \right\|_F = \left\| \frac{1}{2}\big(- Y_{k+1} + \sqrt{Y_{k+1}^\top Y_{k+1} + \frac{4}{\lambda} I}\big) - \frac{1}{2}\big(- Y_\lambda + \sqrt{Y_\lambda^\top Y_\lambda + \frac{4}{\lambda} I}\big) \right\|_F \tag{3.63}$$

$$= \frac{1}{2} \left\| -(Y_{k+1} - Y_\lambda) + (\sqrt{Y_{k+1}^\top Y_{k+1} + \frac{4}{\lambda} I} - \sqrt{Y_\lambda^\top Y_\lambda + \frac{4}{\lambda} I}) \right\|_F \tag{3.64}$$

$$= \frac{1}{2} \left\| -(Y_{k+1} - Y_\lambda) + (A(Y_{k+1}) - A(Y_\lambda)) \right\|_F \tag{3.65}$$

$$\leq \frac{1}{2} \left\| Y_{k+1} - Y_\lambda \right\|_F + \frac{1}{2} \left\| A(Y_{k+1}) - A(Y_\lambda) \right\|_F , \tag{3.66}$$

where $Y_{k+1} = \frac{1}{\lambda}\widehat{\Sigma} - Z_k^{\text{AM}}$ and $Y_\lambda = \frac{1}{\lambda}\widehat{\Sigma} - \widehat{Z}_\lambda$. The last derivation step makes use of the triangle inequality. Using lemma (3), we have,

$$\left\| \Theta_{k+1}^{\text{AM}} - \widehat{\Theta}_\lambda \right\|_F \leq \frac{1}{2} \left\| Y_{k+1} - Y_\lambda \right\|_F + \frac{1}{2} \alpha_\lambda \left\| Y_{k+1} - Y_\lambda \right\|_F . \tag{3.67}$$

Therefore

$$\left\| \Theta_{k+1}^{\text{AM}} - \widehat{\Theta}_\lambda \right\|_F \leq C_\lambda \left\| Y_{k+1} - Y_\lambda \right\|_F = C_\lambda \left\| Z_k^{\text{AM}} - \widehat{Z}_\lambda \right\|_F , \tag{3.68}$$

where

$$C_\lambda = \frac{1}{2} + \frac{1}{2}\alpha_\lambda = 1 - \frac{1}{4}(\frac{\lambda}{4}\Lambda_{max}(Y_{k+1})^2 + 1)^{-1/2}(\frac{\lambda}{4}\Lambda_{max}(Y_\lambda)^2 + 1)^{-1/2} \tag{3.69}$$

$$= 1 - (\lambda\Lambda_{max}(Y_{k+1})^2 + 4)^{-1/2}(\lambda\Lambda_{max}(Y_\lambda)^2 + 4)^{-1/2} \leq 1, \tag{3.70}$$

$\Lambda_{max}(X)$ is the largest eigenvalue of X in absolute value. The rest is to show that both

$\Lambda_{max}(Y_\lambda)$ and $\Lambda_{max}(Y_{k+1})$ are bounded using assumption (2). For $\Lambda_{max}(Y_{k+1})$, we have,

$$\Lambda_{max}(Y_{k+1}) = \|Y_{k+1}\|_2 = \left\|\frac{1}{\lambda}\widehat{\Sigma} - Z_k^{\text{AM}}\right\|_2 \tag{3.71}$$

$$\leq \left\|\frac{1}{\lambda}\widehat{\Sigma}\right\|_2 + \left\|Z_k^{\text{AM}}\right\|_2 \tag{3.72}$$

$$\leq \frac{1}{\lambda}c_{\widehat{\Sigma}} + \left\|Z_k^{\text{AM}} - \widehat{Z}_\lambda\right\|_F + \left\|\widehat{Z}_\lambda\right\|_F. \tag{3.73}$$

Combining (3.62) and (3.68), we have,

$$\left\|Z_{k+1}^{\text{AM}} - \widehat{Z}_\lambda\right\|_F \leq \left\|\Theta_{k+1}^{\text{AM}} - \widehat{\Theta}_\lambda\right\|_F \leq C_\lambda \left\|Z_k^{\text{AM}} - \widehat{Z}_\lambda\right\|_F. \tag{3.74}$$

Therefore,

$$\left\|Z_k^{\text{AM}} - \widehat{Z}_\lambda\right\|_F \leq \left\|Z_{k-1}^{\text{AM}} - \widehat{Z}_\lambda\right\|_F \leq \cdots \leq \left\|Z_0^{\text{AM}} - \widehat{Z}_\lambda\right\|_F. \tag{3.75}$$

Continuing with (3.73), we have,

$$\Lambda_{max}(Y_{k+1}) \leq \frac{1}{\lambda}c_{\widehat{\Sigma}} + \left\|Z_k^{\text{AM}} - \widehat{Z}_\lambda\right\|_F + \left\|\widehat{Z}_\lambda\right\|_F \tag{3.76}$$

$$\leq \frac{1}{\lambda}c_{\widehat{\Sigma}} + \left\|Z_0^{\text{AM}} - \widehat{Z}_\lambda\right\|_F + \left\|\widehat{Z}_\lambda\right\|_F \tag{3.77}$$

$$\leq \frac{1}{\lambda}c_{\widehat{\Sigma}} + \left\|Z_0^{\text{AM}}\right\|_F + 2\left\|\widehat{Z}_\lambda\right\|_F. \tag{3.78}$$

Since $\widehat{Z}_\lambda$ is the minimizer of a strongly convex function, its norm is bounded. And we also have $\left\|Z_0^{\text{AM}}\right\|_F$ bounded in Algorithm (3), so $\Lambda_{max}(Y_{k+1})$ is bounded above whenever $\lambda < \infty$. For $\Lambda_{max}(Y_\lambda)$, we have,

$$\Lambda_{max}(Y_\lambda) = \left\|\frac{1}{\lambda}\widehat{\Sigma} - \widehat{Z}_\lambda\right\|_2 \tag{3.79}$$

$$\leq \left\|\frac{1}{\lambda}\widehat{\Sigma}\right\|_2 + \left\|\widehat{Z}_\lambda\right\|_2 \tag{3.80}$$

$$\leq \frac{1}{\lambda}c_{\widehat{\Sigma}} + \left\|\widehat{Z}_\lambda\right\|_2. \tag{3.81}$$

Therefore both $\Lambda_{max}(Y_\lambda)$ and $\Lambda_{max}(Y_{k+1})$ are bounded in (3.70), *i.e.* $0 < C_\lambda < 1$ is a constant only depending on λ.

□

Theorem 1. *Under the assumptions 1 & 2, if $\rho \asymp \sqrt{\frac{\log d}{m}}$, where ρ is the l_1 penalty, d is the dimension of problem and m is the number of samples, the Alternate Minimization algorithm*

has linear convergence rate for optimization objective defined in (3.6). The k^{th} iteration of the AM algorithm satisfies,

$$\left\|\Theta_k^{AM} - \Theta^*\right\|_F \leqslant C_\lambda \left\|\Theta_{k-1}^{AM} - \widehat{\Theta}_\lambda\right\|_F + \mathcal{O}_{\mathbb{P}}\left(\sqrt{\frac{(\log d)/m}{\min(\frac{1}{(d+s)}, \frac{\lambda}{d^2})}}\right), \tag{3.20}$$

where $0 < C_\lambda < 1$ is a constant depending on λ.

Proof. *(1) Error between $\widehat{\Theta}_\lambda$ and $\widehat{\Theta}_\mathcal{G}$*

Combining the following two equations:

$$f(\widehat{\Theta}_\lambda, \widehat{Z}_\lambda; \rho, \lambda) = \min_{\Theta, Z} f(\Theta, Z; \rho, \lambda) \leqslant \min_{\Theta} f(\Theta, Z = \Theta; \rho, \lambda) = \min_{\Theta} \mathcal{G}(\Theta; \rho) = \mathcal{G}(\widehat{\Theta}_\mathcal{G}; \rho),$$

$$f(\Theta, Z; \rho, \lambda) = \mathcal{G}(\Theta; \rho) + \rho(\|Z\|_1 - \|\Theta\|_1) + \frac{1}{2}\lambda \|Z - \Theta\|_F^2,$$

we have:

$$0 \leqslant \mathcal{G}(\widehat{\Theta}_\lambda; \rho) - \mathcal{G}(\widehat{\Theta}_\mathcal{G}; \rho) \leqslant \rho(\left\|\widehat{\Theta}_\lambda\right\|_1 - \left\|\widehat{Z}_\lambda\right\|_1).$$

Note that by the optimality condition, $\nabla_z f(\widehat{\Theta}_\lambda, \widehat{Z}_\lambda, \rho, \lambda) = 0$, we have the fixed point equation

$$\widehat{Z}_\lambda = \eta_{\rho/\lambda}(\widehat{\Theta}_\lambda).$$

Therefore, $\left\|\widehat{\Theta}_\lambda\right\|_1 - \left\|\widehat{Z}_\lambda\right\|_1 \leqslant \frac{\rho d^2}{\lambda}$ and we have:

$$0 \leqslant \mathcal{G}(\widehat{\Theta}_\lambda; \rho) - \mathcal{G}(\widehat{\Theta}_\mathcal{G}; \rho) \leqslant \frac{\rho^2 d^2}{\lambda}. \tag{3.82}$$

Since $\mathcal{G}$ is $\sigma_\mathcal{G}$-strongly convex, where $\sigma_\mathcal{G}$ is independent of the sample covariance matrix $\widehat{\Sigma^*}$ as the hessian of $\mathcal{G}$ is independent of $\widehat{\Sigma}^*$.

$$\frac{\sigma_\mathcal{G}}{2}\left\|\widehat{\Theta}_\lambda - \widehat{\Theta}_\mathcal{G}\right\|_F^2 + \langle \nabla\mathcal{G}(\widehat{\Theta}_\mathcal{G}; \rho), \widehat{\Theta}_\lambda - \widehat{\Theta}_\mathcal{G}\rangle \leqslant \mathcal{G}(\widehat{\Theta}_\lambda; \rho) - \mathcal{G}(\widehat{\Theta}_\mathcal{G}; \rho). \tag{3.83}$$

Therefore,

$$\left\|\widehat{\Theta}_\lambda - \widehat{\Theta}_\mathcal{G}\right\|_F \leqslant \sqrt{\frac{2\rho^2 d^2}{\lambda\sigma_\mathcal{G}}} = \rho d\sqrt{\frac{2}{\lambda\sigma_\mathcal{G}}} = O\left(\rho d\sqrt{\frac{1}{\lambda}}\right) \tag{3.84}$$

□

Proof. *(2) Error between $\widehat{\Theta}_\mathcal{G}$ and Θ^**

Corollary 5 (Theorem 1. of [68]). *Let $\widehat{\Theta}_{\mathcal{G}}$ be the minimizer for the optimization objective $\mathcal{G}(\Theta;\rho)$. Under Assumptions 1 & 2, if $\rho \asymp \sqrt{\frac{\log d}{m}}$,*

$$\left\|\widehat{\Theta}_{\mathcal{G}} - \Theta^*\right\|_F = \mathcal{O}_{\mathbb{P}}\left(\sqrt{\frac{(d+s)\log d}{m}}\right) \tag{3.85}$$

□

(3) Error between Θ_k^{AM} and Θ^*

Under the conditions in Corollary 5, we use triangle inequality to combine the above results with Corollary 5 and Lemma 4.

$$\begin{aligned}
\left\|\Theta_k^{\text{AM}} - \Theta^*\right\|_F &\leqslant \left\|\Theta_k^{\text{AM}} - \widehat{\Theta}_\lambda\right\|_F + \left\|\widehat{\Theta}_\lambda - \widehat{\Theta}_{\mathcal{G}}\right\|_F + \left\|\widehat{\Theta}_{\mathcal{G}} - \Theta^*\right\|_F && (3.86)\\
&\leqslant C_\lambda \left\|\Theta_{k-1}^{\text{AM}} - \widehat{\Theta}_\lambda\right\|_F + O\left(\rho d\sqrt{\frac{1}{\lambda}}\right) + \mathcal{O}_{\mathbb{P}}\left(\sqrt{\frac{(d+s)\log d}{m}}\right) && (3.87)\\
&\leqslant C_\lambda \left\|\Theta_{k-1}^{\text{AM}} - \widehat{\Theta}_\lambda\right\|_F + \mathcal{O}_{\mathbb{P}}\left(\sqrt{\frac{(d+s)\log d}{m.\min(1, \frac{(d+s)\lambda}{d^2})}}\right) && (3.88)\\
&\leqslant C_\lambda \left\|\Theta_{k-1}^{\text{AM}} - \widehat{\Theta}_\lambda\right\|_F + \mathcal{O}_{\mathbb{P}}\left(\sqrt{\frac{(\log d)/m}{\min(\frac{1}{(d+s)}, \frac{\lambda}{d^2})}}\right) && (3.89)
\end{aligned}$$

3.4.2 Implications of linear convergence of AM on `GLAD`

From the theorem 1, one can see that by optimizing the quadratic penalty parameter λ, one can adjust the C_λ in the bound. We observe that at each stage k, an optimal penalty parameter λ_k can be chosen depending on the most updated value C_λ. An adaptive sequence of penalty parameters $(\lambda_1, \ldots, \lambda_K)$ should achieve a better error bound compared to a fixed λ. Since C_λ is a very complicated function of λ, the optimal λ_k is hard to choose manually.

Besides, the linear convergence guarantee in this theorem is based on the sparse regularity parameter $\rho \asymp \sqrt{\frac{\log d}{m}}$. However, choosing a good ρ value in practice is tedious task as shown in our experiments.

In summary, the implications of this theorem are:

- An adaptive sequence $(\lambda_1, \ldots, \lambda_K)$ should lead to an algorithm with better convergence than a fixed λ, but the sequence may not be easy to choose manually.

- Both ρ and the optimal λ_k depend on the corresponding error $\|\Theta^{\text{AM}} - \widehat{\Theta}_\lambda\|_F$, which make these parameters hard to prescribe manually.
- Since, the AM algorithm has a fast linear convergence rate, we can run it for a fixed number of iterations K and still converge with a reasonable error margin.

Our learning augmented deep architecture, GLAD, can tune these sequence of λ_k and ρ parameters jointly using gradient descent. Moreover, we refer to a recent work by [67] where they considered minimizing the graphical lasso objective with a general nonconvex penalty. They showed that by iteratively solving a sequence of adaptive convex programs one can achieve even better error margins (refer their Algorithm 1 & Theorem 3.5). In every iteration they chose an adaptive regularization matrix based on the most recent solution and the choice of nonconvex penalty. We thus hypothesize that we can further improve our error margin if we make the penalty parameter ρ nonconvex and problem dependent function. We choose ρ as a function depending on the most up-to-date solution $(\Theta_k, \widehat{\Sigma}, Z_k)$, and allow different regularizations for different entries of the precision matrix. Such flexibility potentially improves the ability of GLAD model to recover the sparse graph.

3.5 Scaling for large matrices

We have shown in our experiments that we can train GLAD on smaller number of nodes and get reasonable results for recovering graph structure with considerably larger nodes (See the following experiments section). In this section, we focus on scaling up on the inference/test part.

With the current GPU implementation, we can can handle around 10,000 nodes for inference. For problem sizes with more than 100,000 nodes, we propose to use the randomized algorithm techniques given in [69]. Kindly note that scaling up GLAD is our ongoing work and we just present here one of the directions that we are exploring. The approach presented below is to give some rough idea and may contain loose ends.

Randomized algorithms techniques are explained elaborately in [69]. Specifically, we

will use some of their key results

- P1. (Theorem 2.1) We will use the length-squared sampling technique to come up with low-rank approximations
- P2. (Theorem 2.5) For any large matrix $A \in R^{m \times n}$, we can use approximate it as $A \approx CUR$, where $C \in R^{m \times r}, U \in R^{s \times r}, R \in R^{r \times m}$.
- P3. (Section 2.3) For any large matrix $A \in R^{m \times n}$, we can get its approximate SVD by using the property $E(R^T R) = A^T A$ where R is a matrix obtained by length-squared sampling of the rows of matrix A.

The steps for doing approximate AM updates, i.e. of equations(3.16, 3.17). Using property P3, we can approximate $Y^T Y \approx R^T R$.

$$\sqrt{Y^\top Y + \tfrac{4}{\lambda} I} \approx \sqrt{R^T R + \tfrac{4}{\lambda} I} \approx V \sqrt{\left(\Sigma^2 + \tfrac{4}{\lambda}\right)} V^T \tag{3.90}$$

where V is the right singular vectors of R. Thus, we can combine this approximation with the sketch matrix approximation of $Y \approx CUR$ to calculate the update in equation(3.16). Equation(3.17) is just a thresholding operation and can be done efficiently with careful implementation. We are looking in to the experimental as well as theoretical aspects of this approach.

We are also exploring an efficient distributed algorithm for `GLAD`. We are investigating into parallel MPI based algorithms for this task (`https://stanford.edu/~boyd/admm.html` is a good reference point). We leverage the fact that the size of learned neural networks are very small, so that we can duplicate them over all the processors. This is also an interesting future research direction.

3.6 Experiments

In this section, we report several experiments to compare `GLAD` with traditional algorithms and other data-driven algorithms. The results validate the list of desiderata mentioned previously. Especially, it shows the potential of pushing the boundary of tradi-

tional graph recovery algorithms by utilizing data. Python implementation (tested on P100 GPU) is available[1]. Exact experimental settings details are provided in this section for the thorough reader. **Evaluation metric.** We use normalized mean square error (NMSE) and probability of success (PS) to evaluate the algorithm performance. NMSE is $10\log_{10}(\mathbb{E}\left\|\Theta^p - \Theta^*\right\|_F^2 / \mathbb{E}\left\|\Theta^*\right\|_F^2)$ and PS is the probability of correct signed edge-set recovery, i.e., $\mathbb{P}\left[\text{sign}(\Theta^p_{ij}) = \text{sign}(\Theta^*_{ij}), \forall (i,j) \in \mathbf{E}(\Theta^*)\right]$, where $\mathbf{E}(\Theta^*)$ is the true edge set. **Notation.** In all reported results, D stands for dimension d of the random variable, M stands for sample size and N stands for the number of graphs (precision matrices) that is used for training.

3.6.1 Synthetic Dataset generation

For sections 3.6.2 and 3.6.4, the synthetic data was generated based on the procedure described in [65]. A d dimensional precision matrix Θ was generated by initializing a $d \times d$ matrix with its off-diagonal entries sampled i.i.d. from a uniform distribution $\Theta_{ij} \sim \mathcal{U}(-1, 1)$. These entries were then set to zero based on the sparsity pattern of the corresponding Erdos-Renyi random graph with a certain probability p. Finally, an appropriate multiple of the identity matrix was added to the current matrix, so that the resulting matrix had the smallest eigenvalue as 1. In this way, Θ was ensured to be a well-conditioned, sparse and positive definite matrix. This matrix was then used in the multivariate Gaussian distribution $\mathcal{N}(0, \Theta^{-1})$, to obtain M i.i.d samples.

3.6.2 Benefit of data-driven gradient-based algorithm

Inconsistent optimization objective. Traditional algorithms are typically designed to optimize the ℓ_1-penalized log likelihood. Since it is a convex optimization, convergence to optimal solution is usually guaranteed. However, this optimization objective is different from the true error. Taking ADMM as an example, it is revealed in Fig. 3.2 that, although

[1] code: https://drive.google.com/open?id=16POE4TMp7UUieLcLqRzSTqzkVHm2stlM

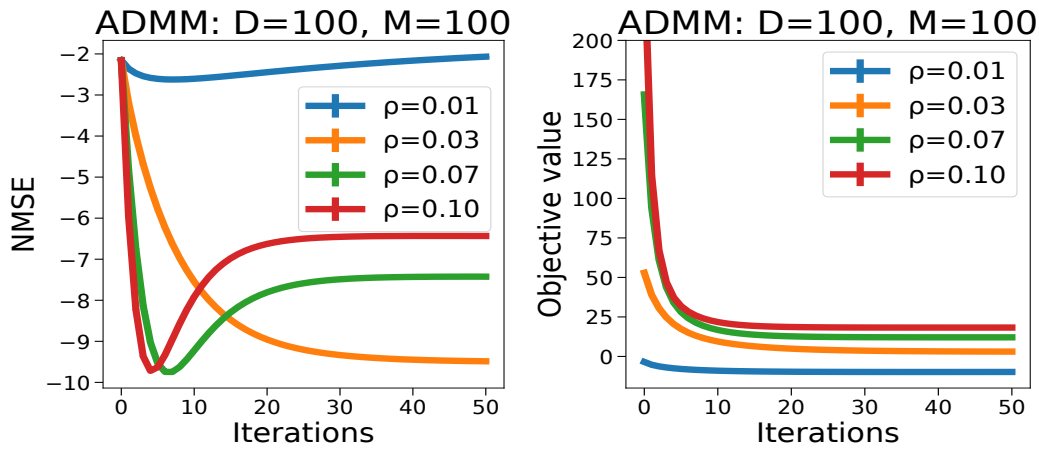

Figure 3.2: Convergence of ADMM in terms of NMSE and optimization objective. The plots are for the ADMM method on the Erdos-Renyi graphs (fixed sparsity $p = 0.1$) with dimension $D = 100$ and number of samples $M = 100$. The results are averaged over 100 test graphs with 10 sample batches per graph. The std-err = $\sigma/\sqrt{1000}$ is shown. Refer Section 3.6.1 for more details on data generation process.

Table 3.1: NMSE results for ADMM.

ρ \ λ	5	1	0.5	**0.1**	0.01
0.01	-2.51	-2.25	-2.06	-2.06	-2.69
0.03	-5.59	-9.05	9.48	**-9.61**	-9.41
0.07	-9.53	-7.58	-7.42	-7.38	-7.46
0.1	-9.38	-6.51	-6.43	-6.41	-6.50
0.2	-6.76	-4.68	-4.55	-4.47	-4.80

the optimization objective always converges, errors of recovering true precision matrices measured by NMSE have very different behaviors given different regularity parameter ρ, which indicates the necessity of directly optimizing NMSE and hyperparameter tuning.

Expensive hyperparameter tuning. Although hyperparameters of traditional algorithms can be tuned if the true precision matrices are provided as a validation dataset, we want to emphasize that hyperparamter tuning by **grid search** is a tedious and hard task.

Table(3.1) shows the final NMSE values for the ADMM method on the random graph (fixed sparsity $p = 0.1$) with dimension $D = 100$ and number of samples $M = 100$. We fixed the initialization parameter of Θ_0 as $t = 0.1$ and chose appropriate update rate α for λ . It is important to note that the NMSE values are very sensitive to the choice of t as well. These parameter values changed substantially for a new problem setting.Table 3.1 shows

that the NMSE values are very sensitive to both ρ and the quadratic penalty λ of ADMM method. For instance, the optimal NMSE in this table is -9.61 when $\lambda = 0.1$ and $\rho = 0.03$. However, it will increase by a large amount to -2.06 if ρ is only changed slightly to 0.01. There are many other similar observations in this table, where slight changes in parameters can lead to significant NMSE differences, which in turns makes grid-search very expensive. G-ISTA and BCD follow similar trends.

Fig. 3.1 shows the average NMSE values over 100 test graphs obtained by the ADMM algorithm on the synthetic data for dimension $D = 100$ and $M = 100$ samples as we vary the values of penalty parameter ρ and lagrangian parameter λ. The offset parameter for Θ_0 was set to $t = 0.1$. The NMSE values are very sensitive to the choice of t as well. These parameter values changes substantially for a new problem setting. G-ISTA and BCD follow similar trends.

Refer Fig. 3.3 for instance. These plots highlights the hyperparameter sensitivity of the traditional methods for model selection consistency experiments.For a fair comparison against `GLAD` which is data-driven, in all following experiments, all hyperparameters in traditional algorithms are **fine-tuned** using validation datasets, for which we spent extensive efforts. In contrast, the gradient-based training of `GLAD` turns out to be much easier.

3.6.3 `GLAD`: Architecture details

`GLAD` parameter settings (refer Fig. 3.5): ρ_{nn} was a 4 layer neural network and Λ_{nn} was a 2 layer neural network. Both used 3 hidden units in each layer. The non-linearity used for hidden layers was $\tanh$, while the final layer had sigmoid (σ) as the non-linearity for both, ρ_{nn} and Λ_{nn}. The learnable offset parameter of initial Θ_0 was set to $t = 1$. It was unrolled for $L = 30$ iterations. The learning rates were chosen to be around $[0.01, 0.1]$ and multi-step LR scheduler was used. The optimizer used was 'adam'. The best nmse model was selected based on the validation data performance. Fig. 3.4 explores the performance of GLAD on using varying number of unrolled iterations L.

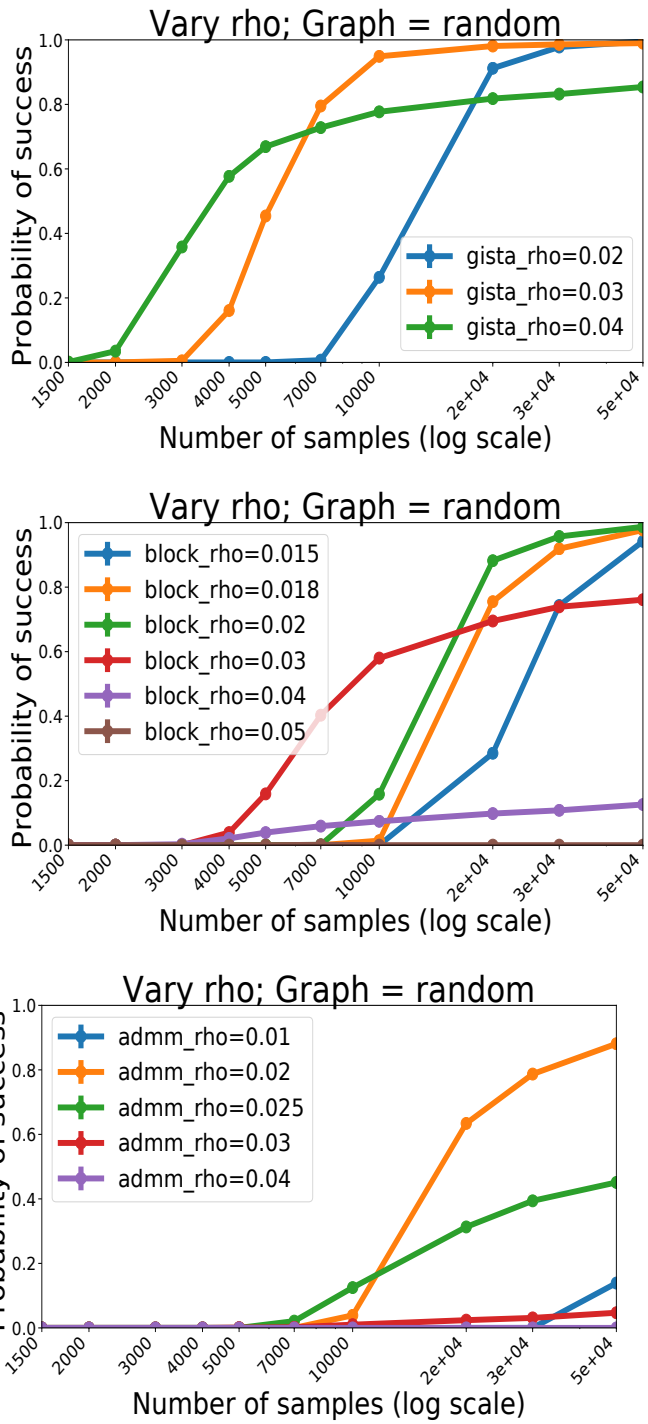

Figure 3.3: We attempt to illustrate how the traditional methods are very sensitive to the hyperparameters and it is a tedious exercise to finetune them. The problem setting is same as described in Section 3.6.1. For all the 3 methods shown above, we have already tuned the algorithm specific parameters to a reasonable setting. Now, we vary the L_1 penalty term ρ and can observe that how sensitive the probability of success is with even slight change of ρ values.

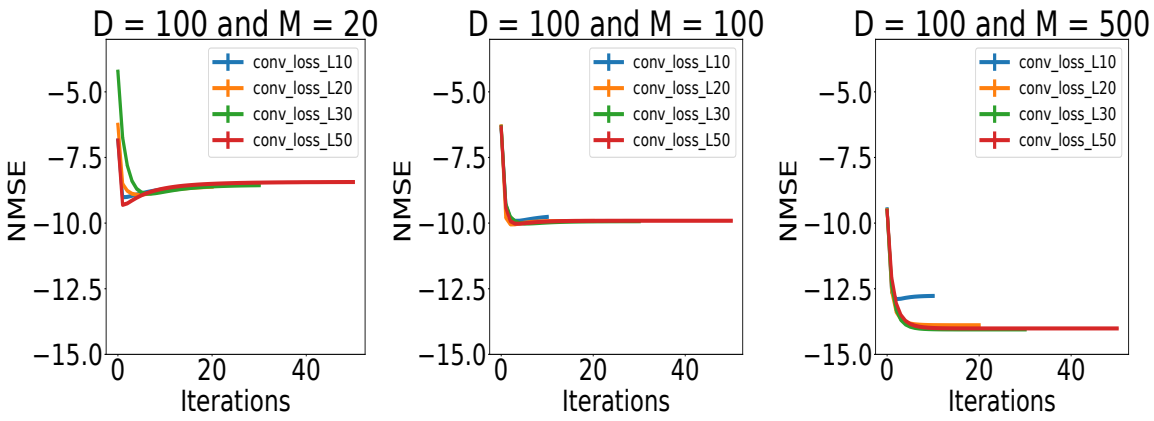

Figure 3.4: Varying the number of unrolled iterations. The results are averaged over 1000 test graphs. The L variable is the number of unrolled iterations. We observe that the higher number of unrolled iterations better is the performance.

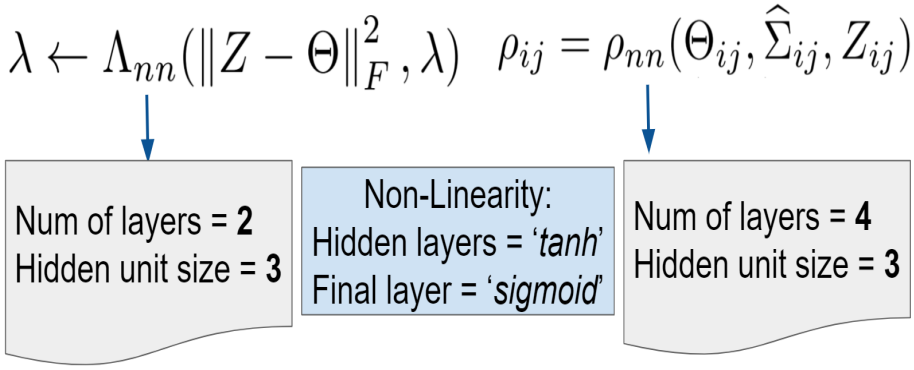

Figure 3.5: Minimalist neural network architectures designed for GLAD experiments in Sections(3.6.4, 3.6.5, 3.6.6, 3.7.1).

3.6.4 Convergence

We follow the experimental setting in [65, 70, 71] to generate data and perform synthetic experiments on multivariate Gaussians. Each off-diagonal entry of the precision matrix is drawn from a uniform distribution, i.e., $\Theta^*_{ij} \sim \mathcal{U}(-1, 1)$, and then set to zero with probability $p = 1 - s$, where s means the sparsity level. Finally, an appropriate multiple of the identity matrix was added to the current matrix, so that the resulting matrix had the smallest eigenvalue as 1 (refer to Section 3.6.1). We use 30 unrolled steps for GLAD (Fig. 3.5) and compare it to G-ISTA, ADMM and BCD. All algorithms are trained/finetuned using 10 randomly generated graphs and tested over 100 graphs.

Table 3.2: Millisecond (ms) per iteration for different methods.

Time/itr	D=25	D=100
ADMM	1.45	16.45
G-ISTA	37.51	41.47
GLAD	2.81	20.23

Convergence results and average runtime of different algorithms on Nvidia's P100 GPUs are shown in Fig. 3.6 and Table 3.2 respectively. `GLAD` consistently converges faster and gives lower NMSE. Although the fine-tuned G-ISTA also has decent performance, the computation time in each iteration is much longer than `GLAD` because it requires line search steps. Besides, we could also see a progressive improvement of `GLAD` across its iterations.

Fig. 3.6 shows the NMSE comparison plots for fixed sparsity and mixed sparsity synthetic Erdos-renyi graphs. The dimension was fixed to $D = 100$ and the number of samples vary as $M = [20, 100, 500]$. The top row has the sparsity probability $p = 0.5$ for the Erdos-Renyi random graph, whereas for the bottom row plots, the sparsity probabilities are uniformly sampled from $\sim \mathcal{U}(0.05, 0.15)$. For finetuning the traditional algorithms, a validation dataset of 10 graphs was used. For the GLAD algorithm, 10 training graphs were randomly chosen and the same validation set was used.

3.6.5 Recovery probability

As analyzed by [64], the recovery guarantee (such as in terms of Frobenius norm) of the ℓ_1 regularized log-determinant optimization significantly depends on the sample size and other conditions. Our `GLAD` directly optimizes the recovery objective based on data, and it has the potential of pushing the sample complexity limit. We experimented with this and found the results positive.

We follow [64] to conduct experiments on GRID graphs, which satisfy the conditions required in [64]. Furthermore, we conduct a more challenging task of recovering restricted but randomly constructed graphs. The probability of success (PS) is non-zero only if the

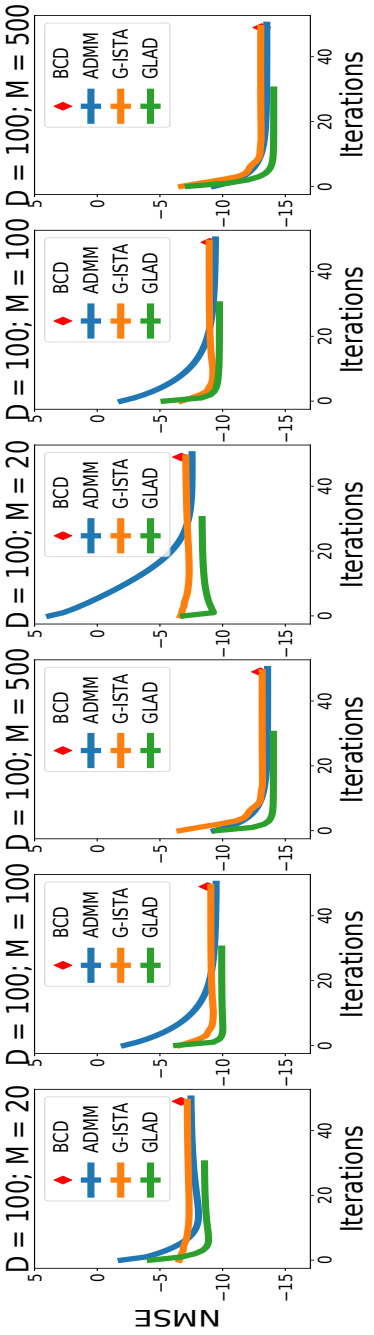

Figure 3.6: GLAD vs traditional methods. *Left 3 plots:* Sparsity level is fixed as $s = 0.1$. *Right 3 plots:* Sparsity level of each graph is randomly sampled as $s \sim \mathcal{U}(0.05, 0.15)$. Results are averaged over 100 test graphs where each graph is estimated 10 times using 10 different sample batches of size M. Standard error is plotted but not visible. Intermediate steps of BCD are not evaluated because we use sklearn package[72] and can only access the final output. Section 3.6.3 explains the `GLAD` architecture.

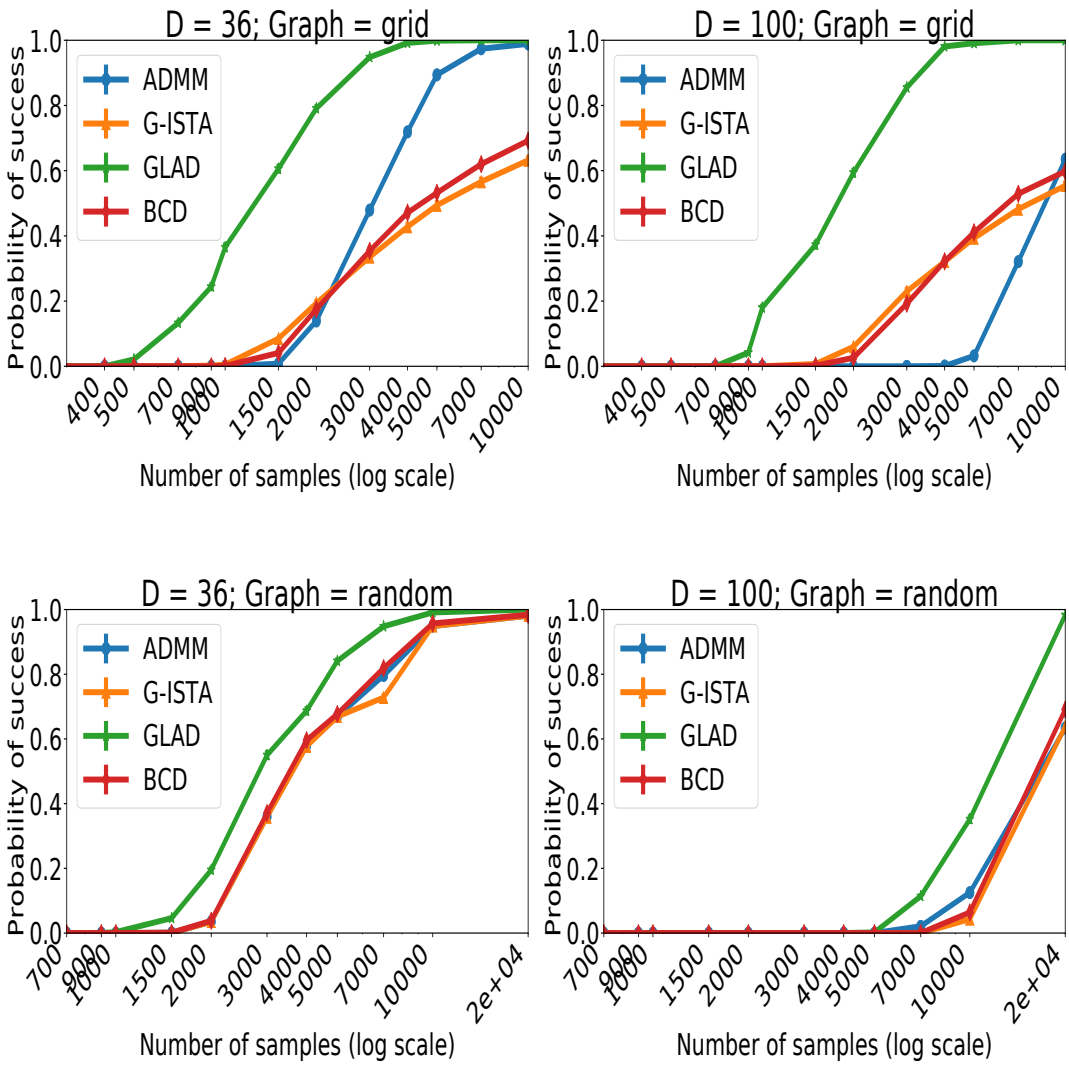

Figure 3.7: Sample complexity for model selection consistency.

algorithm recovers all the edges with correct signs, plotted in Fig. 3.7. GLAD consistently outperforms traditional methods in terms of sample complexity as it recovers the true edges with considerably fewer number of samples.

Details for experiments in Fig. 3.7. Two different graph types were chosen for this experiment which were inspired from [64]. In the 'grid' graph setting, the edge weight for different precision matrices were uniformly sampled from $w \sim \mathcal{U}(0.12, 0.25)$. The edges within a graph carried equal weights. The other setting was more general, where the graph was a random Erdos-Renyi graph with probability of an edge was $p = 0.05$. The off-diagonal entries of the precision matrix were sampled uniformly from $\sim \mathcal{U}[0.1, 0.4]$. The

Table 3.3: AUC on 100 test graphs for D=39: For experiment settings, refer Table 1 of [58]. Gaussian Random graphs with sparsity $p = 0.05$ were chosen and edge values sampled from $\sim \mathcal{U}(-1, 1)$.

Methods	M=15	M=35	M=100
BCD	0.578±0.006	0.639±0.007	0.704±0.006
DG-39	0.664±0.008	0.738±0.006	0.759±0.006
DG-39+P	0.672±0.008	0.740±0.007	0.771±0.006
GLAD	**0.788±0.003**	**0.811±0.003**	**0.878±0.003**

parameter settings for GLAD were the same as described in Section 3.6.3. The model with the best PS performance on the validation dataset was selected. train/valid/test=10/10/100 graphs were used with 10 sample batches per graph.

3.6.6 Data Efficiency

Having a good inductive bias makes GLAD's architecture quite data-efficient compared to other deep learning models. For instance, the state-of-the-art 'DeepGraph' by [58] is based on CNNs. It contains orders of magnitude more parameters than GLAD. Furthermore, it takes roughly $100,000$ samples, and several hours for training their DG-39 model. In contrast, GLAD learns well with less than 25 parameters, within 100 training samples, and notably less training time. Table 3.3 also shows that GLAD significantly outperforms DG-39 model in terms of AUC (Area under the ROC curve) by just using 100 training graphs, typically the case for real world settings. Fully connected DL models are unable to learn from such small data and hence are skipped in the comparison.

Table(3.3) shows AUC (with std-err) comparisons with the DeepGraph model. For the experiment settings, refer Table 1 of [58]. Gaussian Random graphs with sparsity $p = 0.05$ were chosen and edge values sampled from $\sim \mathcal{U}(-1, 1)$. GLAD was trained on only 10 graphs with 5 sample batches per graph. The dimension of the problem is $D = 39$. The architecture parameter choices of GLAD were the same as described in Section 3.6.3 and it performs consistently better along all the settings by a significant AUC margin.

3.7 SynTReN gene expression simulator details

The SynTReN [24] is a synthetic gene expression data generator specifically designed for analyzing the structure learning algorithms. The topological characteristics of the synthetically generated networks closely resemble the characteristics of real transcriptional networks. The generator models different types of biological interactions and produces biologically plausible synthetic gene expression data enabling the development of data-driven approaches to recover the underlying network.

The SynTReN simulator details for Section 3.7.1. For performance evaluation, a connected Erdos-Renyi graph was generated with probability as $p = 0.05$. The precision matrix entries were sampled from $\Theta_{ij} \sim \mathcal{U}(0.1, 0.2)$ and the minimum eigenvalue was adjusted to 1 by adding an appropriate multiple of identity matrix. The SynTReN simulator then generated samples from these graphs by incorporating biological noises, correlation noises and other input noises. All these noise levels were sampled uniformly from $\sim \mathcal{U}(0.01, 0.1)$. The Fig. 3.8 shows the NMSE comparisons for a fixed dimension $D = 25$ and varying number of samples $M = [10, 25, 100]$. The number of training/validation graphs were set to 20/20 and the results are reported on 100 test graphs. In these experiments, only 1 batch of M samples were taken per graph to better mimic the real world setting.

Fig. 3.11 visualizes the edge-recovery performance of the above trained `GLAD` models on a subnetwork of true Ecoli bacteria data.which contains 30 edges and $D = 43$ nodes. The Ecoli subnetwork graph was fed to the SynTReN simulator and M samples were obtained. SynTReN's noise levels were set to 0.05 and the precision matrix edge values were set to $w = 0.15$. For the `GLAD` models, the training was done on the same settings as the gene-data NMSE plots with $D = 25$ and on corresponding number of samples M. Though, `GLAD` was trained on lower dimensional graphs $D = 25$, it was able to generalize reasonably to higher dimensional graph $D = 43$. The Chapter 4 further analysis such generalization criteria in more details for real data settings.

3.7.1 Gene regulation data

The SynTReN [24] is a synthetic gene expression data generator specifically designed for analyzing the sparse graph recovery algorithms. It models different types of biological interactions and produces biologically plausible synthetic gene expression data. Fig. 3.8 shows that GLAD performs favourably for structure recovery in terms of NMSE on the gene expression data. As the governing equations of the underlying distribution of the SynTReN are unknown, these experiments also emphasize the ability of GLAD to handle non-Gaussian data.

Fig. 3.11 visualizes the edge-recovery performance of GLAD models trained on a sub-network of true Ecoli bacteria data. We denote, TPR: True Positive Rate, FPR: False Positive Rate, FDR: False Discovery Rate. The number of simulated training/validation graphs were set to 20/20. One batch of M samples were taken per graph (details in the previous Section 3.7). Although, GLAD was trained on graphs with $D = 25$, it was able to robustly recover a higher dimensional graph $D = 43$ structure.

3.7.2 Results on real data

Section 3.7.2 contains details of the experiments done on real E.Coli data. The GLAD model was trained using the SynTReN simulator. We use the real data from the 'DREAM 5 Network Inference challenge' [73]. This dataset contains 3 compendia that were obtained from microorganisms, some of which are pathogens of clinical relevance. Each compendium consists of hundreds of microarray experiments, which include a wide range of genetic, drug, and environmental perturbations. We test our method for recovering the true E.coli network from the gene expression values recorded by doing actual microarray experiments.

The E.coli dataset contains 4511 genes and 805 associated microarray experiments. The true underlying network has 2066 discovered edges and 150214 pairs of nodes do not have an edge between them. There is no data about the remaining edges. For our experiments, we only consider the discovered edges as the ground truth, following the challenge data

Table 3.4: GLAD vs other methods for the DREAM network inference challenge real E.Coli data.

Methods	BCD	GISTA	**GLAD**
AUC	0.548	0.541	**0.572**

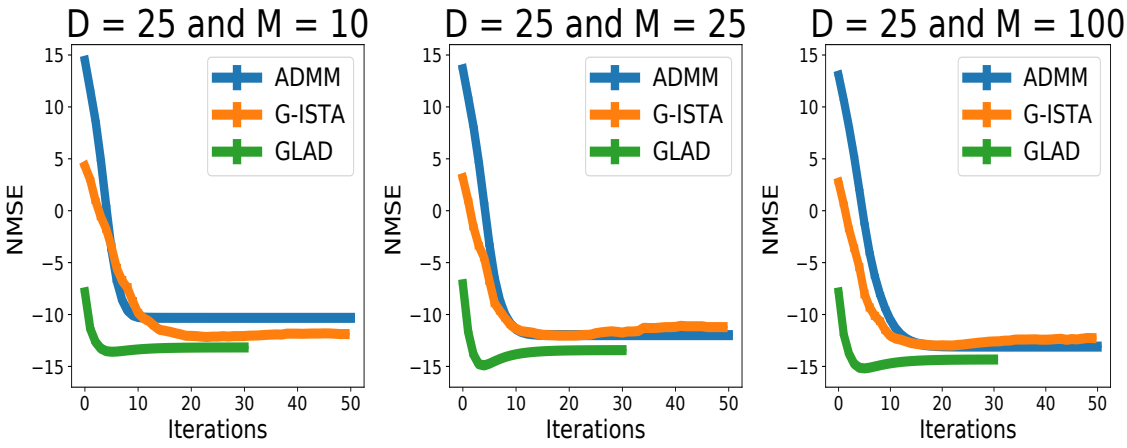

Figure 3.8: Performance on the SynTReN generated gene expression data with graph as Erdos-renyi having sparsity $p = 0.05$. Refer Section 3.7 for experiment details.

settings. We remove the genes that have zero degree and then we get a subset of 1081 genes. For our predictions, we ignore the direction of the edges and only consider retrieving the connections between genes.

We train the GLAD model using the SynTReN simulator on the similar settings as described in Section 3.7. Briefly, GLAD model was trained on D=50 node graphs sampled from Erdos-Renyi graph with sparsity probability $\sim U(0.01, 0.1)$, noise levels of SynTReN simulator sampled from $\sim U(0.01, 0.1)$ and $\Theta_{ij} \sim U(0.1, 0.2)$). The model was unrolled for 15 iterations. This experiment also evaluates GLAD's ability to generalize to different distribution from training as well as scaling ability to more number of nodes.

We report the AUC scores for E.coli network in Table 3.4 . We can see that GLAD improves over the other competing methods in terms of Area Under the ROC curve (AUC). We understand that it is challenging to model real datasets due to the presence of many unknown latent extrinsic factors, but we do observe an advantage of using data-driven parameterized algorithm approaches.

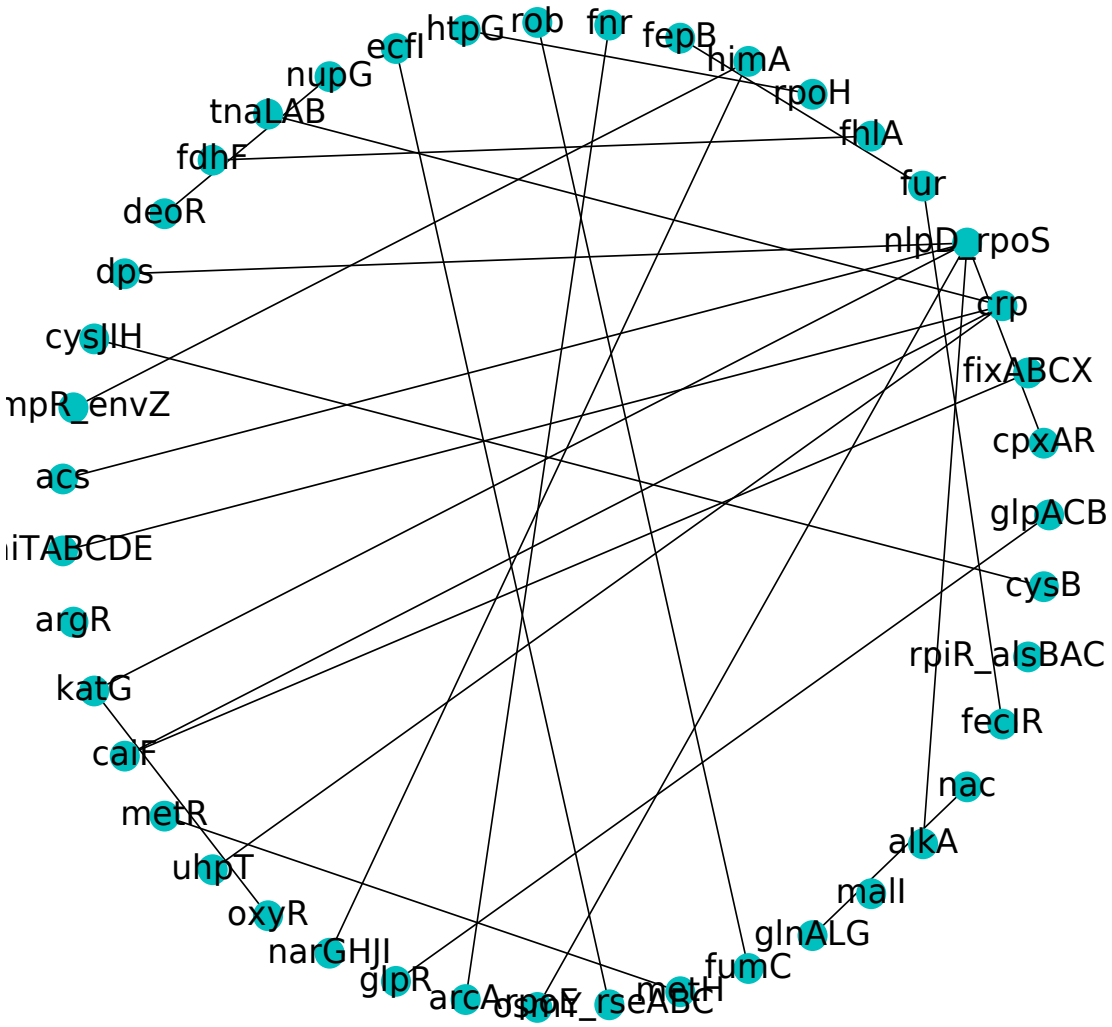

Figure 3.9: True graph for a sub-network of the *E. coli* consisting of 43 genes and 30 interactions.

3.8 Discussions on alternate designs for unrolled algorithms

3.8.1 Comparison with ADMM optimization based unrolled algorithm

In order to find the best unrolled architecture for sparse graph recovery, we considered many different optimization techniques and came up with their equivalent unrolled neural network based deep model. In this section, we compare with the closest unrolled deep model based on ADMM optimization, (`ADMMu`), and analyze how it compares to `GLAD`. Section 3.8.2

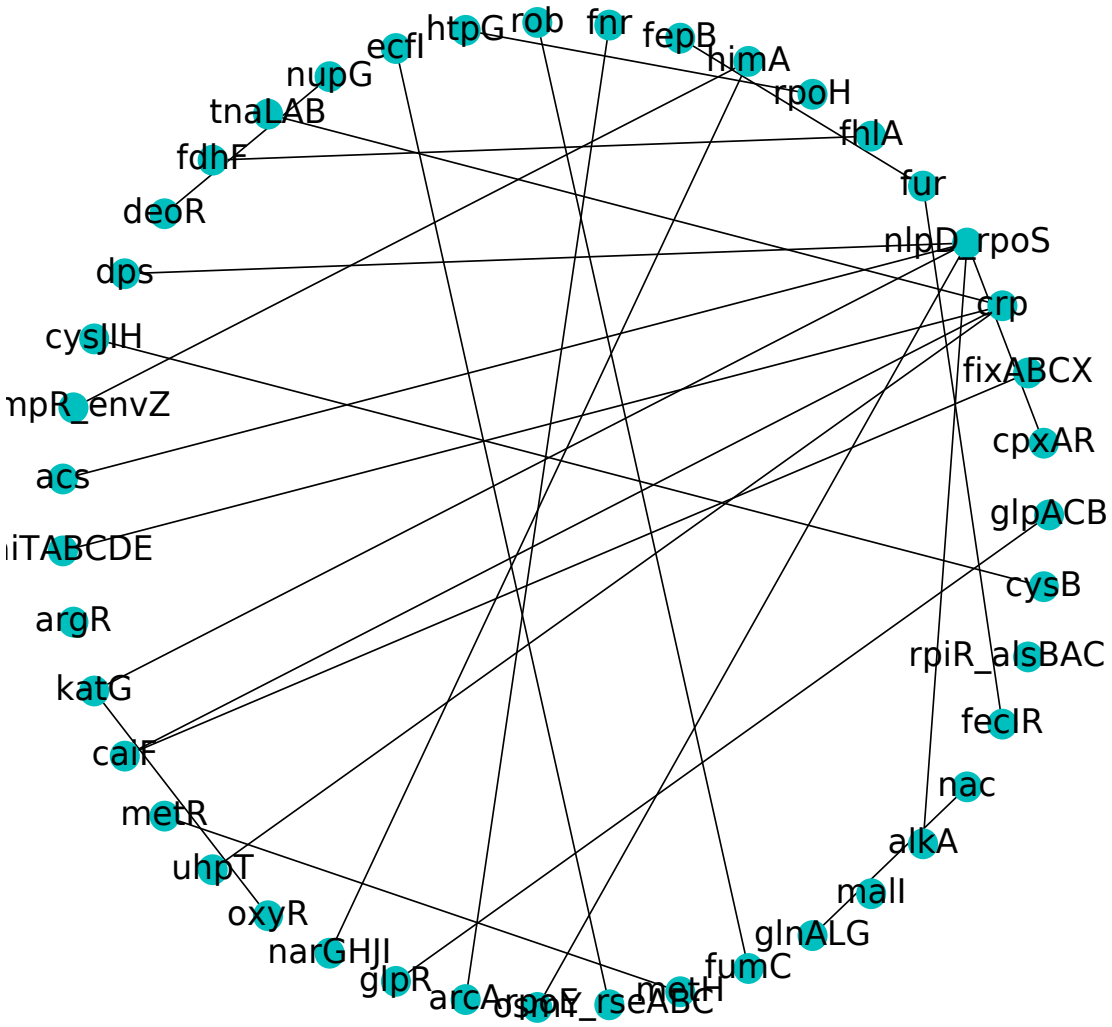

Figure 3.10: Recovered graph by GLAD: M=10, fdr=0.613, tpr=0.913, fpr=0.114. Note that the number of samples $M = 10$ is less than the dimension of the problem $D = 43$.

lists down further such techniques for future exploration.

Unrolled model for ADMM: Algorithm 4 describes the unrolled model ADMMu updates. ρ_{nn} was a 4 layer neural network and Λ_{nn} was a 2 layer neural network. Both used 3 hidden units in each layer. The non-linearity used for hidden layers was $\tanh$, while the final layer had sigmoid (σ) as the non-linearity for both ,ρ_{nn} and Λ_{nn}. The learnable offset parameter of initial Θ_0 was set to $t = 1$. It was unrolled for $L = 30$ iterations. The learning rates were chosen to be around $[0.01, 0.1]$ and multi-step LR scheduler was used. The optimizer used

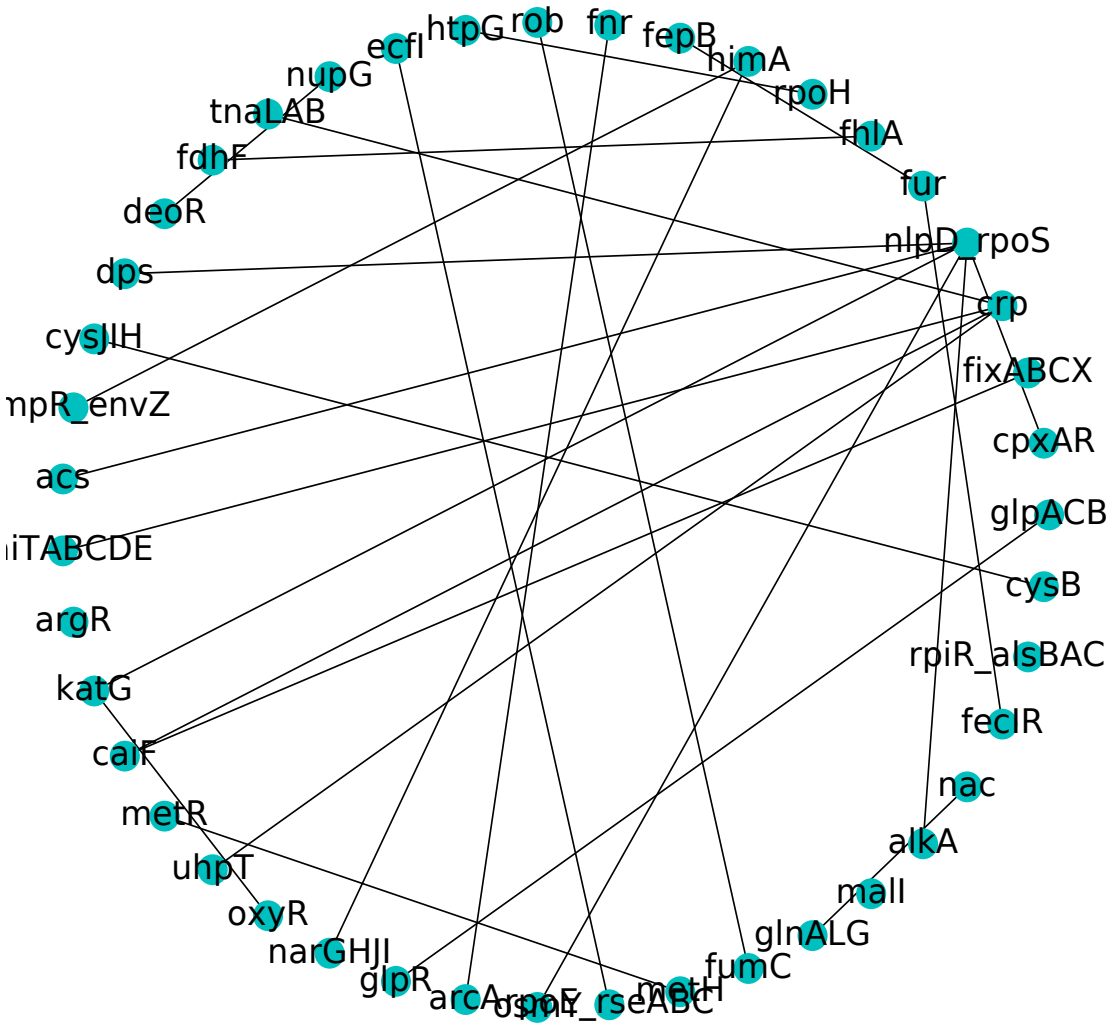

Figure 3.11: Recovered graph by GLAD (Refer Fig. 3.9&3.10): M=100, fdr=0.236, tpr=0.986, fpr=0.024. Note that the number of samples $M = 100$ is more than the dimension of the problem $D = 43$. Increasing the samples reduces the fdr by discovering more true edges.

was 'adam'.

Fig. 3.12 compares GLAD with ADMMu on the convergence performance with respect to synthetically generated data. The settings were kept same as described in Fig. 3.6. As evident from the plots, we see that GLAD consistently performs better than ADMMu. We had similar observations for other set of experiments as well. Hence, we chose AM based unrolled algorithm over ADMM's as it works better empirically and has less parameters.

Algorithm 4: `ADMMu`

Function `ADMMu-cell` $(\widehat{\Sigma}, \Theta, Z, U, \lambda)$ **:**

$\lambda \leftarrow \Lambda_{nn}(\|Z - \Theta\|_F^2, \lambda)$

$Y \leftarrow \lambda^{-1}\widehat{\Sigma} - Z + U$

$\Theta \leftarrow \frac{1}{2}\big(-Y + \sqrt{Y^\top Y + \frac{4}{\lambda}I}\big)$

For *all* i, j **do**

$\rho_{ij} = \rho_{nn}(\Theta_{ij}, \widehat{\Sigma}_{ij}, Z_{ij}, \lambda)$

$Z_{ij} \leftarrow \eta_{\rho_{ij}}(\Theta_{ij} + U_{ij})$

$U \leftarrow U + \Theta - Z$

return Θ, Z, U, λ

Function `ADMMu` $(\widehat{\Sigma})$ **:**

$\Theta_0 \leftarrow (\widehat{\Sigma} + tI)^{-1}$, $\lambda_0 \leftarrow 1$

For $k = 0$ *to* $K - 1$ **do**

$\Theta_{k+1}, Z_{k+1}, U_{k+1}, \lambda_{k+1}$

$\leftarrow$`ADMMu-cell` $(\widehat{\Sigma}, \Theta_k, Z_k, U_k, \lambda_k)$

return Θ_K, Z_K

Although, we are not entirely confident but we hypothesize the reason for above observations as follows. In the ADMM update equations (3.4 & 3.5), both the Lagrangian term and the penalty term are intuitively working together as a 'function' to update the entries Θ_{ij}, Z_{ij}. Observe that U_k can be absorbed into Z_k and/or Θ_k and we expect our neural networks to capture this relation. We thus expect `GLAD` to work at least as good as `ADMMu`. In our formulation of unrolled `ADMMu` (Algorithm 4) the update step of U is not controlled by neural networks (as the number of parameters needed will be substantially larger) which might be the reason of it not performing as well as `GLAD`.

Our empirical evaluations corroborate this logic that just by using the penalty term we can maintain all the desired properties and learn the problem dependent 'functions' with a small neural network.

3.8.2 Different designs tried for data-driven algorithm

We tried multiple unrolled parameterizations of the optimization techniques used for solving the graphical lasso problem which worked to varying levels of success. We list here a

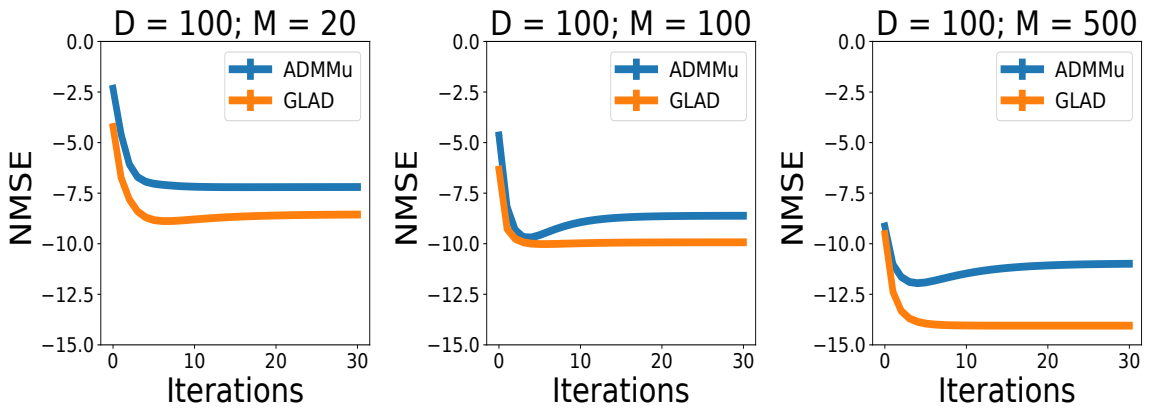

Figure 3.12: Convergence on Erdos-random graphs with fixed sparsity ($p = 0.1$). All the settings are same as the fixed sparsity case described in Figure 3.6. We see that the AM based parameterization '`GLAD`' consistently performs better than the ADMM based unrolled architecture '`ADMMu`'.

few, in interest for helping researchers to further pursue this recent and novel approach of data-driven algorithm designing.

1. ADMM + ALISTA parameterization: The threshold update for Z_{k+1}^{AM} can be replaced by ALISTA network [21]. The stage I of ALISTA is determining W, which is trivial in our case as $D = I$. So, we get $W = I$. Thus, combining ALISTA updates along with AM's we get an interesting unrolled algorithm for our optimization problem.
2. G-ISTA parameterization: We parameterized the line search hyperparameter c as well as replaced the next step size determination step by a problem dependent neural network of Algorithm(1) in [65]. The main challenge with this parameterization is to main the PSD property of the intermediate matrices obtained. Learning appropriate parameterization of line search hyperparameter such that PSD condition is maintained remains an interesting aspect to investigate.
3. Mirror Descent Net: We get a similar set of update equations for the graphical lasso optimization. We identify some learnable parameters, use neural networks to make them problem dependent and train them end-to-end.
4. For all these methods we also tried unrolling the neural network as well. In our experience we found that the performance does not improve much but the convergence becomes

unstable.

3.9 Conclusion & Future work

We presented a novel neural network, GLAD, for the sparse graph recovery problem based on an unrolled Alternating Minimization algorithm. We theoretically prove the linear convergence of AM algorithm as well as empirically show that learning can further improve the sparse graph recovery. The learned GLAD model is able to push the sample complexity limits thereby highlighting the potential of using algorithms as inductive biases for deep learning architectures. Further development of theory is needed to fully understand and realize the potential of this new direction.

CHAPTER 4
AN UNROLLED DEEP LEARNING FRAMEWORK FOR SINGLE CELL GENE REGULATORY NETWORKS

4.1 Introduction

Inferring gene regulatory networks (GRNs) from microarray or RNA-seq gene expression data sets has been an active area of research for more than two decades. This topic is receiving renewed attention in the context of single-cell transcriptomic data [74, 75, 23]. In contrast to bulk transcriptome data of prior years, single cell RNA-Sequencing (scRNA-Seq) data provides cellular level activity, albeit with higher levels of noise and more data sparsity [76]. Several GRN reconstruction methods which were originally developed for bulk transcriptional data have been applied [77, 78, 79, 80] or adapted [81, 82, 83, 84] to single-cell data, and new methods have been developed specifically for it [85, 86, 87, 88, 89, 90]. For a recent review on the performance and comparative evaluation of various GRN methods for single-cell transcriptomic data, see [23].

An important class of methods developed for GRN inference is based on unsupervised learning. GRNBoost2 [80] and GENIE3 [91], among the top performing methods [74, 23], operate by fitting regression functions between the expression values of the transcription factors (TFs) and other genes. An alternate approach is to pose GRN inference as a graphical lasso problem with l_1 regularization [66]. All these approaches are primarily unsupervised in nature. Recently, simulators which generate synthetic scRNA-Seq data guided by GRNs have progressed significantly [25, 23]. They can generate realistic data by modeling sources of variation in single cell RNA-Seq data such as noise intrinsic to the process of transcription, extrinsic variation indicative of different cell states, technical variation, and measurement noise and bias. These simulators have been primarily developed and applied to systematically

benchmark GRN inference methods.

In this work, we propose to leverage GRN-guided simulators in a novel way to enable supervised learning of GRNs from scRNA-Seq data. Motivated by the recent successes in supervised neural network (NN) based algorithms in learning graphical models [58] and the `GLAD` work from Chapter 3, we propose a deep learning (DL) framework that takes expression data as input and outputs the corresponding GRN. For the purpose of supervised training of our framework, we use the SERGIO [25] simulator to generate a corpus of training examples containing gene-expression datasets and the corresponding GRNs.

Our DL framework consists of two novel modeling choices. Firstly, we leverage the expressive ability of NNs to capture the dependencies between TFs and the corresponding genes they regulate, by aptly using an NN in a multi-task learning framework. Secondly, in order to capture sparsity of the GRNs observed in the real world, we design an unrolled algorithm for our framework. Unrolled algorithm is an emerging paradigm in machine learning that is gaining prominence in discovering sparse graphical models. Key advantages include (a) fewer parameters to be learned, (b) less supervised data points required for training, (c) comparable or better performance than state-of-the-art methods, and (d) more interpretability. Unrolled algorithms have been successfully designed in recent works, for example, RNA secondary structure prediction method `E2EFold` [92] and sparse graph recovery technique `GLAD` described in Chapter 3. Both the multi-task learning NN and sparsity related parameters are optimized jointly in our DL framework using supervision.

Our unrolled DL model, termed `GRNUlar` (pronounced "granular") for Gene Regulatory Network Unrolled algorithm, outperforms state-of-the-art methods on both simulated data and real data including from human and mouse. Our learned neural model is comparably more robust to high levels of noise often observed in single-cell expression data. We demonstrate that our methods benefit from the supervision obtained through synthetic data simulators. To the best of our knowledge, this work constitutes the first unrolled deep learning framework for GRN inference, and the first application of simulators for training

neural algorithms for doing GRN inference for scRNA-Seq data.

4.2 Problem Setting and Challenges

We consider the input gene expression data to have D genes and M samples, $X \in \mathbb{R}^{M \times D}$. Let $\mathcal{G} = [1, D]$ be the set of genes and $\mathcal{T} \subset \mathcal{G}$ be those which are transcription factors (TFs). We aim to identify directed interactions of the form (t, g), where $t \in \mathcal{T}$ and $g \in \mathcal{G}$. Note that there can be interactions between TFs themselves. For our method, we assign directed edges between the TFs and other genes and the interactions between TFs are represented by undirected edges. We thus output completed partially directed acyclic graphs (PDAGs), which represent equivalence classes of DAGs [93].

Existing approaches: The common approach followed by many state-of-the-art methods for GRN inference is based on fitting regression functions between the expression values of TFs and each of the genes. Usually, a sparsity constraint is also associated with the regression function to identify the top influencing TFs for every gene.

Generally, the objective function used for GRN recovery in various methods is a variant of the equation given below. For all $g \in \mathcal{G}$,

$$X_g = f_g(X_{\mathcal{T}}) + \epsilon \tag{4.1}$$

Eq. 4.1 can be viewed as fitting a regression between the expression value of each gene as a function of the TFs and some random noise. The simplest model will be to assume that the function f_g is linear. One of the widely used methods, TIGRESS [94], assumes a linear function of the following form for every gene: $f_g(X_{\mathcal{T}}) = \sum_{t \in \mathcal{T}} \beta_{t,g} X_t$. Another top performing method, GENIE3 [91], assumes each f_g to be a random forest. GRNBoost2 [80] further uses gradient boosting techniques over the GENIE3 architecture to do efficient GRN reconstruction. Supervised learning methods for recovering the GRN by inferring the gene-gene interactions for all the possible gene pairs have also been recently explored [95, 96]. Our approach differs from them as we formulate our framework to jointly optimize for

all the interactions.

Drawbacks: There are two major drawbacks in current approaches. The first is in choosing the function f_g, which can be improved further to better capture non-linear relations and make the method more robust to noise in data. The second is tuning the sparsity related hyperparameter for the GRN which usually requires an additional posthoc scoring step. Such scoring process to obtain the desired sparsity of GRN is sub-optimal. A better approach would be to jointly optimize the sparsity along with discovering the underlying GRN.

4.3 The Proposed `GRNUlar` Framework

To overcome the aforementioned drawbacks, we propose a DL framework with the following three key components:

1. *Choice of f_g:* We model f_g using neural networks which are able to learn expressive class of functions [1]. We view the NN itself as a multi-task learning framework which can be easily interpreted as the underlying GRN, where the multiple tasks correspond to the inference of TFs for multiple genes.

2. *Use supervision:* We develop a deep learning model which leverages simulators to generate training examples for supervised learning. The training data consists of multiple input gene expression datasets and the corresponding GRNs. We hypothesize that tuning GRN recovery models under this supervision will lead us to better capture intricacies of real data, and potentially improve upon the unsupervised methods.

3. *Capture sparsity:* We use the recently developed unrolled algorithm paradigm to design the deep architecture which can model the underlying sparsity as a parameter that can be learned under supervision.

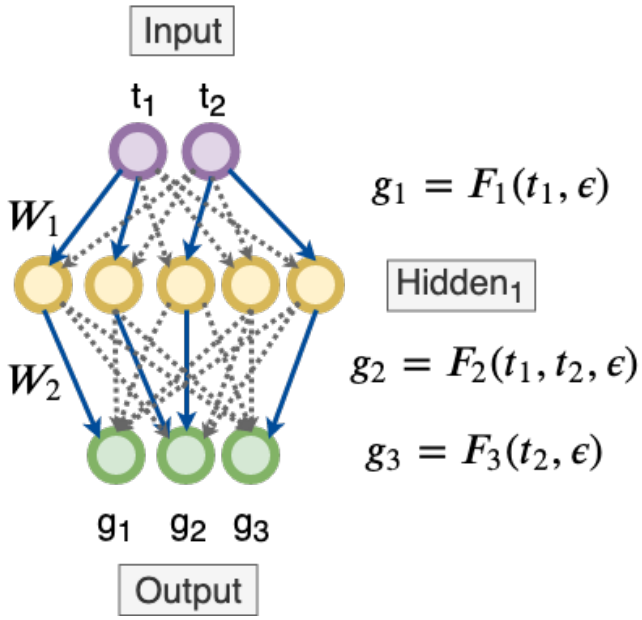

Figure 4.1: We start with a fully connected NN indicating all genes are dependent on all the input TFs (dotted black lines). Assume that in the process of discovering the underlying sparse GRN our algorithm zeroes out all the edge weights except the 'blue' ones. Now, if there is a path from an input TF to an output gene, then we conclude that the output gene is dependent on the corresponding input TF.

4.3.1 Neural network modeling of regression functions

Neural networks are capable of representing rich classes of highly non-linear functions. We combine the regression formulation in Eq. 4.1 with neural networks in a multi-task learning framework [97] (Fig. 4.1) to learn multiple non-linear regression functions estimating dependencies between each gene and the set of transcription factors. If there is a path in the NN from an input TF to the output gene, then the output gene is dependent on the corresponding TF. We thus want sparse NN weights $\mathcal{W} = \{W_1, W_2, \cdots, W_C\}$ in order to obtain a sparse graph. We can easily obtain the dependency matrix between input TFs and output genes as a matrix multiplication, where $\Theta_p = \Pi_i |W_i| = |W_1| \times |W_2| \times \cdots \times |W_C|$ represents the matrix product of the neural network weights. This dependency matrix can be directly interpreted as the underlying GRN. Non-zero values in the matrix Θ_p correspond to edges between the corresponding pairs of genes.

This multi-task neural network architecture is superior to boosted decision tree based

formulations as it is more expressive, and does not need the additional posthoc scoring step. It is also more expressive than a simple non-linear model with additive noise because the NN is jointly optimizing the regression for all the output genes (multi-task learning) and this helps it capture the common dependencies between the TFs and output genes. This also makes the NN model more robust towards external noises as jointly optimizing for all the gene expression values will mitigate the effect of any expression value anomalies that seeped in during the experiments.

Designing the unrolled algorithm

Many neural network representations can satisfy Eq. 4.1, which leads to multiple possible GRNs. These GRNs will vary mostly in terms of the sparsity obtained and it is hard to tune for the desired sparsity of the GRN manually. Unrolled algorithms help resolve this problem as the sparsity related hyperparameters (eg., the weight of the l_1 norm term) can be learned from supervision. Our aim is to get a deep model which optimizes the multi-task learning NN along with learning the optimal sparsity pattern using the supervision provided from simulator data. We follow similar procedure as the unrolled algorithm designed for sparse graph recovery by using the Alternative Minimization (AM) algorithm which we introduced in the Chapter 3.

Identifying the inductive bias: We wish to simultaneously fit the regression formulations in Eq. 4.1 as well as learn the desired sparsity of the underlying GRN. One way to achieve this is to jointly optimize the regression error with an l_1 penalty term over the dependency matrix. Thus, we consider the following non-linear optimization objective function for the regression with l_1 penalty

$$\arg\min_{\mathcal{W}} \sum_{k=1}^{M} \left\| X_{\mathcal{G}}^{k} - f_{\mathcal{W}}(X_{\mathcal{T}}^{k}) \right\|^2 + \rho \left\| \Pi_i |W_i| \right\|_1 \tag{4.2}$$

where $f_{\mathcal{W}}(X_{\mathcal{T}}^{k})$ is a neural network. Note, $X_{\mathcal{T}}^{k}$ represents the expression values for all the $\mathcal{T}$ TFs and $X_{\mathcal{G}}^{k}$ represents the expression values for all the $\mathcal{G}$ genes for the k^{th} sample or experiment. For example, we can define a 2 layer neural network with 'ReLU' non-linearity

as $f_{W_1,W_2}(X_{\mathcal{T}}^k) = W_2 \cdot \text{ReLU}(W_1 \cdot X_{\mathcal{T}}^k + b_1) + b_2$. We learn the weights $\{W_i\}$ and the biases $\{b_i\}$ while optimizing for Eq. 4.2. The dimensions of the weights and biases are chosen such that the neural network input units are equal to '$\mathcal{T}$' TFs and the output units are equal to '$\mathcal{G}$' genes.

Using optimization algorithm as design template: We now identify the iterative updates of a suitable optimization algorithm. Since the objective above is non-linear, we will need an iterative approach to minimize it w.r.t. $\mathcal{W}$. We apply the AM approach to the optimization given in Eq. 4.2. Our problem becomes easier by using AM as we can get closed form solution of the l_1 penalty term. We introduce an additional Lagrange variable Z, such that $Z = \Pi_i |W_i|$ (product of NN weight matrices) and then including the Lagrangian as a square penalty term, we have $\arg\min_{\mathcal{W},Z} \sum_{k=1}^{M} \left\| X_{\mathcal{G}}^k - f_{\mathcal{W}}(X_T^k) \right\|^2 + \rho \left\| Z \right\|_1 + \frac{1}{2}\lambda \left\| \Pi_i |W_i| - Z \right\|_F^2$. Now, alternately minimize Z and Θ for $l \in [0, L]$ iterations as,

$$\mathcal{W}^{(l+1)} \leftarrow \arg\min_{\mathcal{W}} \sum_{k=1}^{M} \left\| X_{\mathcal{G}}^k - f_{\mathcal{W}}(X_T^k) \right\|^2 + \frac{1}{2}\lambda \left\| \Pi_i |W_i| - Z^l \right\|_F^2 \tag{4.3}$$

$$Z_{l+1} \leftarrow \eta_{\rho/\lambda} \left(\Pi_i |W_i^{(l+1)}| \right) \tag{4.4}$$

The update of Z is of the form $f(Z) + \rho \left\| Z \right\|_1$, where $f(Z)$ is a convex function. Similar to [98], the minimizer of this function is the proximal operator given by $\eta_{\rho/\lambda}(\theta) = \text{sign}(\theta) \max(|\theta| - \rho/\lambda, 0)$.

Unrolling the iterations to get deep model: We now unroll the iterative updates to certain iterations, identify key learnable components, and treat the entire unrolled iterations as a single highly-recurrent deep model. We identify the proximal operator $\eta_{\rho/\lambda}$ and λ as the hyperparameters which control the sparsity of the final graph. We now parameterize them using problem dependent NNs as ρ_{nn}, λ_{nn} respectively. These NNs are minimalist in design and take the solution of the previous update to predict the next value, and are learned using supervision. As for Eq. (4.3), we optimize for W_i^l ($\forall i$) by using standard deep learning optimizers. The corresponding values of Z^l can be obtained by plugging in the W_i^l ($\forall i$) in its closed form update, Eq. 4.4. We unroll these updates for L iterations and treat it as a

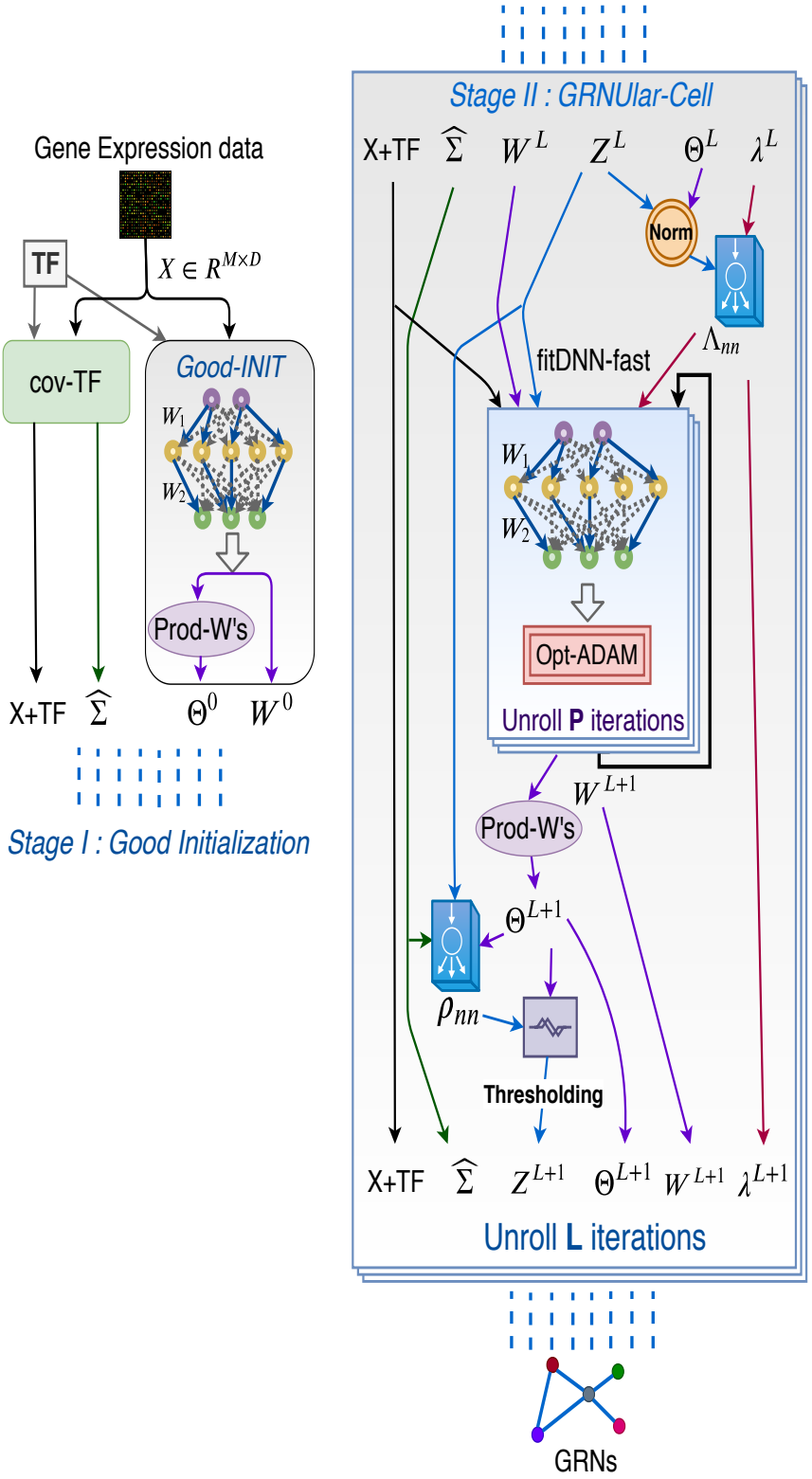

Figure 4.2: Visualizing the `GRNUlar` algorithm's architecture. It is a single deep model which is highly structured & recurrent. It takes gene expression data as input and outputs the corresponding GRN.

Algorithm 5: GRNUlar-basic

Function `covTF` (X, *TFs*) **:**
- $\widehat{\Sigma}_T \leftarrow \frac{1}{M}(X-\mu)^T(X-\mu)$
- Select $\widehat{\Sigma}_T \subset \widehat{\Sigma}$ an $T \times G$ submatrix using the TFs
- **return** $\widehat{\Sigma}_T$

Function `fitDNN` (X, Z, λ, *TFs*) **:**
- Fit $\mathcal{W}$ based on updated regularization terms λ, Z
- $X_T, X_G \leftarrow X$ (using the TFs)
- $f_{\mathcal{W}^0} \leftarrow$ Initialize Neural Networks
- $\mathcal{W} \leftarrow \arg\min_{\mathcal{W}} \sum_{k=1}^{M} \left\| X_G^k - f_{\mathcal{W}}(X_T^k) \right\|^2 + \frac{1}{2}\lambda \left\| \Pi_i |W_i| - Z^l \right\|_F^2$
- (Using standard Deep Learning optimizers for 'E1' epochs)
- $\Theta \leftarrow \Pi_i |W_i|$
- **return** Θ

Function `GRNUlar-cell` ($X, \widehat{\Sigma}_T, \Theta, Z, \lambda$) **:**
- $\lambda \leftarrow \Lambda_{nn}(\|Z-\Theta\|_F^2, \lambda)$
- $\Theta \leftarrow$ `fitDNN`(X, Z, λ)
- **For** *all* i, j **do**
  - $\rho_{ij} = \rho_{nn}(\Theta_{ij}, \widehat{\Sigma}_{T_{ij}}, Z_{ij})$
  - $Z_{ij} \leftarrow \eta_{\rho_{ij}}(\Theta_{ij})$
- **return** Θ, Z, λ

Function `GRNUlar` (X) **:**
- $\lambda^0 \leftarrow 1$
- $\widehat{\Sigma}_T \leftarrow$ `covTF`(X)
- $Z^0 = zeros(T, G)$
- $W_{i's} \leftarrow$ `fitDNN` (X, Z^0, λ^0)
- $\Theta^0 = \Pi_i |W_i^0|$
- **For** $l = 0$ *to* $L-1$ **do**
  - $\Theta^{l+1}, Z^{l+1}, \lambda^{l+1}$
  - $\leftarrow$`GRNUlar-cell` $(X, \widehat{\Sigma}_T, \Theta^l, Z^l, \lambda^l)$
- **return** Θ_L, Z_L

highly structured deep model (Fig. 4.2).

Algorithm 5, `GRNUlar-basic` provides a supervised learning framework for the unrolled model directly based on the updates of the AM algorithm, Eqs. 4.3 & 4.4. We typically require $E1 \sim [200, 400]$ epochs to minimize Eq. 4.3. This slows down the algorithm significantly. To circumvent this issue, we propose a modification to `GRNUlar-basic` algorithm by using a 'good' initialization technique.

We posit that, if we optimize for the 1^{st} term of Eq. 4.3 beforehand and obtain good

Algorithm 6: GRNUlar

Function `covTF` (X, *TFs*) **:**
- $\widehat{\Sigma}_T \leftarrow \frac{1}{M}(X-\mu)^T(X-\mu)$
- Select $\widehat{\Sigma}_T \subset \widehat{\Sigma}$ an $T \times G$ submatrix using the TFs
- **return** $\widehat{\Sigma}_T$

Function `fitDNN-fast` ($X, Z, \mathcal{W}, \lambda$, *TFs*) **:**
- $X_T, X_G \leftarrow X$ (using the TFs)
- **For** $p = 0$ *to* $P-1$ **do**
  - $J^p = \sum_{k=1}^{M} \left\| X_G^k - f_{\mathcal{W}}(X_T^k) \right\|^2 + \frac{\lambda}{2} \left\| \Pi_i |W_i| - Z \right\|_F^2$
  - $\mathcal{W}^{p+1} \leftarrow$ opt-update$(W_{i's}^p, \nabla J^p)$
- $\Theta \leftarrow \Pi_i |W_i|$
- **return** $\Theta, \mathcal{W}$

Function `GRNUlar-cell` ($X, \widehat{\Sigma}_T, \mathcal{W}, \Theta, Z, \lambda$) **:**
- $\lambda \leftarrow \Lambda_{nn}(\|Z - \Theta\|_F^2, \lambda)$
- $\Theta, \mathcal{W} \leftarrow$ `fitDNN-fast`$(X, Z, \mathcal{W}, \lambda)$
- **For** *all* i, j **do**
  - $\rho_{ij} = \rho_{nn}(\Theta_{ij}, \widehat{\Sigma}_{T_{ij}}, Z_{ij})$
  - $Z_{ij} \leftarrow \eta_{\rho_{ij}}(\Theta_{ij})$
- **return** $\mathcal{W}, \Theta, Z, \lambda$

Function `goodINIT` (X, *TFs*) **:**
- $\mathcal{W} \leftarrow \arg\min_{\mathcal{W}} \sum_{k=1}^{M} \left\| X_G^k - f_{\mathcal{W}}(X_T^k) \right\|^2$
- (Using standard DL optimizers for 'E1' epochs)
- $\Theta \leftarrow \Pi_i |W_i|$
- **return** $\Theta, \mathcal{W}$

Function `GRNUlar` (X) **:**
- $\Theta^0, \mathcal{W}^0 \leftarrow$ `goodINIT` (**X**)
- $\lambda_0 \leftarrow 1$
- $\widehat{\Sigma}_T \leftarrow$ `covTF`(X)
- $Z^0 = zeros(T, G)$
- **For** $l = 0$ *to* $L-1$ **do**
  - $\mathcal{W}^{l+1}, \Theta^{l+1}, Z^{l+1}, \lambda^{l+1}$
  - $\leftarrow$`GRNUlar-cell` $(X, \widehat{\Sigma}_T, \mathcal{W}^l, \Theta^l, Z^l, \lambda^l)$
- **return** Θ_L, Z_L

initial values of $\mathcal{W}^0$, we then just need to do minor adjustments to the $\mathcal{W}$ as we update Z and λ. We then just need to unroll the optimization (the new `fitDNN-fast` function) for only few iterations $P \sim \{2, 5, 10\}$. This is a significant reduction in the number of unrolled iterations required over the `fitDNN` function. The `GRNUlar-cell` in Algorithm 6 also does not require large number of unrolled iterations L. The neural network ρ_{nn} is learning

the entrywise thresholding operation and λ_{nn} learns to update its value from norm difference and its previous value. We observe that in every iteration of Algorithm 6 we optimize the unrolled parameters ρ_{nn}, λ_{nn} (tiny NNs) to learn the underlying graph sparsity from the supervision provided. Thus, we want to highlight that the overall training does not require much training data as well as the number of unrolled iterations.

A note on backpropagation of gradients: While taking the $\arg\min$ in the `fitDNN` function given in Algorithm 5, we consider the λ and Z^l as constants. In our PyTorch implementation, we 'detach' these variables from the computational graphs while optimizing for $\mathcal{W}$. Though, ideally we can retain the computational graph while optimizing but then the memory consumption increases considerably. Another important concern is related to the runtime of Algorithm 5. The `fitDNN` function is called L times and it initializes a new neural network each time and minimizes it for the optimization function to a very low error based on the regularization provided by the λ and Z values. We typically require $E1 \sim [200, 400]$ epochs to fit the neural network. This slows down the algorithm significantly. To circumvent this issue, we propose a simple modification to `GRNUlar-basic` algorithm by using a 'good' initialization technique as done for the `GRNUlar` Algorithm 6. We also empirically verify that `GRNUlar` algorithm performs equivalent to `GRNUlar-basic` algorithm with significant runtime improvement.

The `GRNUlar` model can be thought of as having 2 stages, namely (refer Fig. 4.2 & Algorithm 6)

Stage I. We first optimize for the first term in Eq. 4.3 beforehand and obtain 'good' initial values of $\mathcal{W}^0$. Thereafter, only minor adjustments are needed to $\mathcal{W}$ as Z and λ are updated. We then just need to unroll the optimization in the `fitDNN-fast` function for only few iterations $P \sim \{2, 5, 10\}$ during Stage II.

Stage II. After getting the 'good' initialization from Stage I, the data and parameters are passed through the `GRNUlar-cell`. It also does not require large number of unrolled iterations L. The neural network ρ_{nn} is learning the entrywise thresholding operation

and λ_{nn} learns to update its value from norm difference and its previous value (see the 'note' below). For every iteration of `GRNUlar-cell` we optimize the unrolled parameters ρ_{nn}, λ_{nn} (tiny NNs) to learn the underlying graph sparsity from the supervision provided. We highlight here that the overall training does not require much data as well as the number of unrolled iterations.

A **note** on parameterizing the NNs of unrolled algorithms. The general idea of neural network based parameterization: For the `GRNUlar` algorithm, we can further parameterize the optimizer update given in '`fitDNN-fast`' function of Algorithm 6 and learn it from the supervision provided, similar to Λ_{nn}. In our current implementation, we use the 'adam' optimizer. We want to highlight that our technique of parameterization in an unrolled fashion is very generic and can be used for any off-the-shelf optimizer. For instance, consider the example of parameterizing gradient descent optimizer which is realized using the Λ_{nn} update. We just need to define the neural network based parameterization in a way that is more generic then the optimizer's update equation. The neural network based update $\lambda^{t+1} \leftarrow \Lambda_{nn}(\|Z - \Theta\|_F^2, \lambda^t)$ subsumes the standard gradient descent update given by $\lambda^{t+1} \leftarrow \lambda^t - \alpha \|Z - \Theta\|_F^2$.

4.4 Training the GRNUlar Framework

GRNs are typically sparse and so we want our loss function to be robust enough to recover sparse edges. Since there are multiple metrics like precision, recall, F1 score etc. which are commonly used for evaluation of the recovered GRNs, it will be useful to define a loss function which can find a desirable balance between them. For the traditional and unsupervised methods, we do not have a learning component based on the loss function and we can thus not tune the output of the algorithm according to our needs. This is an added advantage to use unrolled algorithms as we can learn the model parameters as per our requirements.

For some applications, we might want to focus on precision We want to design the loss

function such that we can tune the function to adjust focus between precision, recall and other metrics.

To address the above concerns, we develop a differentiable version of the F_β score. Let Θ^p represent our predicted graph (adjacency matrix) and let Θ^* be the true underlying graph. We assume that all the entries of $\Theta \in [0, 1]$. We can write the true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) as follows:

$$\begin{aligned} \text{TP} &=< \Theta^p, \Theta^* >;\ \text{FP} =< \Theta^p, 1 - \Theta^* >; \\ \text{FN} &=< 1 - \Theta^p, \Theta^* >;\ \text{TN} =< 1 - \Theta^p, 1 - \Theta^* >; \end{aligned} \tag{4.5}$$

where $< \cdot, \cdot >$ represents matrix inner product, which is the summation of entry-wise products. Based on the above differentiable representations, we define differentiable F_β score and the corresponding loss function as:

$$\text{F}_\beta = (1 + \beta^2) \cdot \text{TP} / \left((1 + \beta^2) \cdot \text{TP} + \beta^2 \cdot \text{FN} + \text{FP}\right);\ \ \text{L}_{\text{F}_\beta} = 1 - \text{F}_\beta \tag{4.6}$$

A value of $\beta > 1$ weighs recall higher than precision as it emphasizes the FNs. Similarly, having $\beta < 1$ attenuates the influence of FNs and thus weigh recall lower than precision.

The inputs to the loss function is an entry-wise absolute value followed by an entry-wise $\tanh$ function. In some cases, we also do an additional diagonal masking operation, $\Theta^p_m = \text{ma}_{\text{diag}} * \Theta^p$, as we want to ignore the self loops.

We define a loss function between the predicted and true adjacency matrix as the combination of the MSE (or Frobenius norm) loss and the L_{F_β} loss. It is often tricky to jointly optimize and balance between multiple loss functions. Following the loss balancing technique described in [99], we introduce a balancing ratio $r = L_{F_\beta}/L_{\text{mse}}$ which adjusts the scales of both the losses. Note that 'r' is detached from computational graph to facilitate back-propagation of gradients. Note that while implementing this scaling, at every epoch, we calculate 'r' by detaching the losses from the computational graph, so that it is constant. We usually adjust the value of β to tune the algorithm to adjust between 'precision' and 'recall'. For training `GRNUlar`, a collection of input expression data and their corresponding

Algorithm 7: GLAD+TF

Function `mask-TF` (Θ, *TFs*) **:**
- mask = initialize adjacency matrix with zeros
- **For** *n in TFs* **do**
  - mask[n, :] = 1
  - mask[:, n] = 1
- **return** $\Theta_K *$ mask

Function `GLADcell-TF` ($\widehat{\Sigma}, \Theta, Z, \lambda$) **:**
- $\lambda \leftarrow \Lambda_{nn}(\|Z - \Theta\|_F^2, \lambda)$
- $Y \leftarrow \lambda^{-1}\widehat{\Sigma} - Z$
- $\Theta \leftarrow \frac{1}{2}\big(-Y + \sqrt{Y^\top Y + \frac{4}{\lambda}I}\big)$
- **For** *all* i, j **do**
  - $\rho_{ij} = \rho_{nn}(\Theta_{ij}, \widehat{\Sigma}_{ij}, Z_{ij})$
  - $Z_{ij} \leftarrow \eta_{\rho_{ij}}(\Theta_{ij})$
- $Z \leftarrow$ `mask-TF`(Z)
- $\Theta \leftarrow$ `mask-TF`(Θ)
- **return** Θ, Z, λ

Function `GLAD+TF` ($\widehat{\Sigma}$) **:**
- $\Theta_0 \leftarrow (\widehat{\Sigma} + tI)^{-1}$, $\lambda_0 \leftarrow 1$
- **For** $k = 0$ *to* $K - 1$ **do**
  - $\Theta_{k+1}, Z_{k+1}, \lambda_{k+1}$
  - $\leftarrow$`GLADcell-TF` ($\widehat{\Sigma}, \Theta_k, Z_k, \lambda_k$)
- **return** Θ_K, Z_K

ground truth GRNs can be sampled from GRN guided realistic data simulators.

$$Loss = r.\left\|\Pi_i|W_i^{(L)}| - W^*\right\|_F^2 + \mathrm{L}_{\mathrm{F}_\beta}(\tanh\{\Pi_i|W_i^{(L)}|\}, W^*) \tag{4.7}$$

The matrix $W^* \in \{0,1\}^{(O\times T)}$ represents the ground truth network where 1 indicates presence of an edge between (t, o). In order to ensure that the entries of $\Theta^p \in [0, 1]$ we pass it through the $\tanh\{|\Theta^p|\}$ operation. We optimize the loss function over the average of data pairs from simulator so that the learned architecture is able to perform well over a family of problem instances.

4.5 GLAD model & proposed modification for TFs

The GRNUlar algorithm is designed to reconstruct GRNs using the TF information. But, there can be cases where we do not know the underlying TFs. Majority of the existing methods are typically designed to include TF information and their performance in terms of recovery quality and runtimes deteriorate significantly if the TFs are not provided. Our experiments corroborate these statements.

We found an alternative method to slightly mitigate above concerns. We directly use the unrolled model GLAD (from Chapter 3) for the GRN inference problem. This model does not use the TF information for GRN recovery. The GLAD work formulates the sparse graph recovery problem (undirected graphical model) for the GRN recovery as fitting a multivariate Gaussian distribution on the input gene expression data with a l_1 normalization term. It is based on the unrolling the iterations of an Alternate Minimization algorithm for the graphical lasso problem with applications towards sparse graph recovery, refer Algorithm 3.

GLAD+TF model: We modify the architecture of GLAD to include prior information about the TFs if available. We assume that all the edges in the actual GRN have at least one gene which belongs to the list of TFs. We introduce a masking operation at every step of the unrolled iterations as shown in Algorithm 7, which eliminates all the unwanted edges that are between the non-TF nodes.

4.6 Experimental Results

4.6.1 Methods Compared and Evaluation Measures

We use AUROC (Area Under the Receiver Operating Characteristics) and AUPRC (Area Under the Precision Recall Curve) values for evaluation [25, 74] and comparison of various methods. We compared GRNUlar with GRNBoost2 [80], GENIE3 [91], and GLAD (refer Chapter 3). GRNBoost2 and GENIE3 are representative of regression based methods, and are among the top performers for single-cell expression data [23]. We used the Arboreto

package to run these algorithms [80]. We did extensive fine tuning of the hyperparameters for both the methods using the training/valid data and reported results on the test data. 'Method+TF' indicates that TF information was utilized for GRN recovery.

`GLAD`, introduced in Chapter 3, is an unrolled algorithm designed for sparse graph recovery. It is based on unrolling the iterations of an Alternate Minimization algorithm for the graphical lasso problem. It fits a multivariate Gaussian distribution on the input gene expression data with an l_1 normalization term. We modified the `GLAD` algorithm to take into account TF information, called `GLAD+TF`, by using a posthoc masking operation that only retains the edges having at least one node as a TF. We used the standard initialization as recommended by the authors. We chose the number of unrolled iterations $L = \{15, 30\}$. For the `GRNUlar` model, we used the same initialization of the thresholding parameters ρ_{nn}, λ_{nn} as proposed for the `GLAD` model. Now, we need to decide the dimensions of neural network (NN) which fits the regression in the `fitDNN-fast` function. Our general strategy is to have number of hidden layers, $depth \geq 2$, of the NN. We roughly choose the number of hidden units in layer 'j' as $H_j \geq 4 \cdot H_{j-1}$ and we also satisfy TFs $\leq H_1$. We empirically observed that we need around $[200, 500]$ iterations to fit the neural network in the `goodINIT` function. We chose the unroll parameters $L = 15$ and the values of $P = \{5, 10, 20\}$ with NN having 2 or 3 layers.

For both the unrolled methods, we chose two models based on AUPRC and AUROC results on the validation data. We use the scaled loss function (Eq. 4.7) to jointly optimize for MSE and F_β loss. The values of β used in our experiments were chosen from the set $\{0.5, 1, 2, 5\}$. We implemented the unrolled algorithms using PyTorch and ran on Nvidia P100 GPUs. We observed that for these unrolled algorithm based approaches, `GRNUlar` in general outperforms `GLAD+TF`. This is probably due to the difference in the choice of inductive bias for designing their architectures as former is based on the regression based formulation while the architecture of the latter is based on the graphical lasso based formulation.

4.6.2 Details of SERGIO simulator for clean and noisy settings

SERGIO provides a list of parameters to simulate cells from different types of biological processes and gene-expression levels with various amount of intrinsic and technical noise. We simulated cells from multiple steady states. When simulating data with no technical noise (what we refer to as clean data), we set the following parameters: *sampling-state*=15 (determines the number of steps of simulations for each steady state); *noise-param*$\sim U[0.1, 0.3]$ (controls the amount of intrinsic noise); *noise-type*="dpd" (the type of intrinsic noise is Dual Production Decay noise, which is the most complex out of all types provided); We set genes' decay parameter to 1. The parameters required to decide the master regulators' basal production cell rate for all cell types - low expression range of production cell rate $\sim U[0.2, 0.5]$ and high expression range of cell rate $\sim U[0.7, 1]$. We chose $K \sim U[1, 5]$, where 'K' denotes the maximum interaction strength between master regulators and target genes. Positive strength values indicate activating interactions and negative indicates repressive interactions and ± 1 signs are randomly assigned. We added the dropout events which are considered to be a major source of technical noise in real data. Parameters which control the amount of dropouts include *shape* (which was set to 20) and *percentile*, which we varied among the values $q = \{25, 50, 75\}$. Larger q corresponds to higher technical noise. All other parameters were set to default values.

4.6.3 Evaluating GRN inference methods on synthetic data

We conducted an exploratory study to gauge the generalization ability of `GRNUlar` for the GRN inference task. To provide supervision, we used the SERGIO simulator. To create random directed graphs (GRNs) we first decided on the number of TFs or master regulators. Then, we randomly added edges between the TFs and the other genes based on sparsity requirements. Also, we randomly added some edges between the TFs themselves but excluded any self-regulation edges and maintained connectivity of the graph.

The graph is then provided as input to the SERGIO simulator to generate corresponding

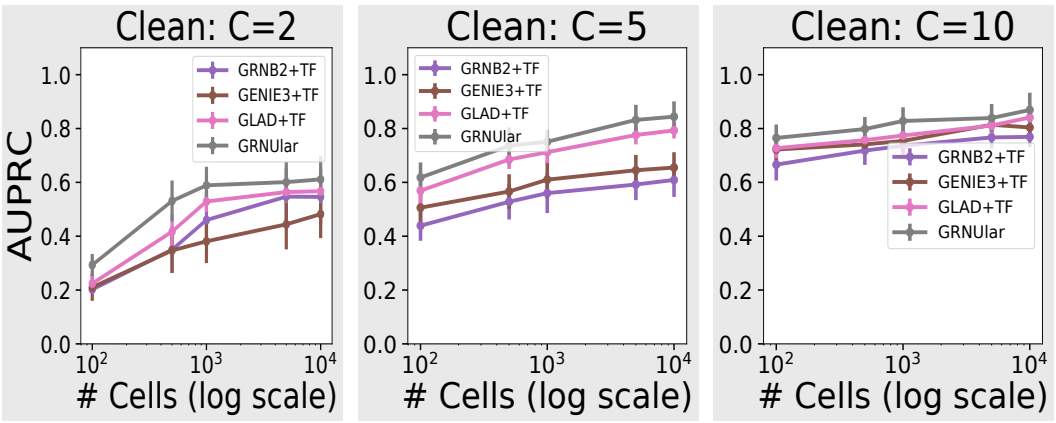

Figure 4.3: Clean data setting of the SERGIO simulator with D=100 genes. As the number of cell types increase from C=2 to C=10, we see that the AUPRC values increase in general. The unrolled algorithms in general outperform the traditional methods.

gene expression data. For the experiments in this subsection, we took train/valid/test = $20/20/50$ graphs respectively, with the number of genes D = 100. All these graphs were sampled from similar settings. We usually choose the ratio of TFs to the total number of genes as 0.1 (~ 10 TFs for D=100) and sparsity of training graphs to be 0.1. We wish to highlight that many graphs are not needed to train the unrolled models as we are primarily learning the sparsity pattern from supervision and need small NNs for the same (refer Fig. 3.5 in Chapter 3).

From the literature on sample complexity theory of sparse graph recovery (eg. [64]) we know that recovery of the underlying graph improves with increasing number of samples. Hence, we ran our experiments with varying number of the total single cells, M = $\{100, 500, 1K, 5K, 10K\}$. We also observed that varying the number of cell types (corresponding to the number of clusters of the cells) of the SERGIO simulator considerably affects GRN inference results, so we also evaluated the methods by varying the number of cell types of the simulator C= $\{2, 5, 10\}$. We adjusted the number of cells per cell type to maintain the same total number of cells. Section 4.6.2 contains detailed description of SERGIO settings. For experiments in this subsection, each data point in the plots represents its value along with standard deviation over the test graphs.

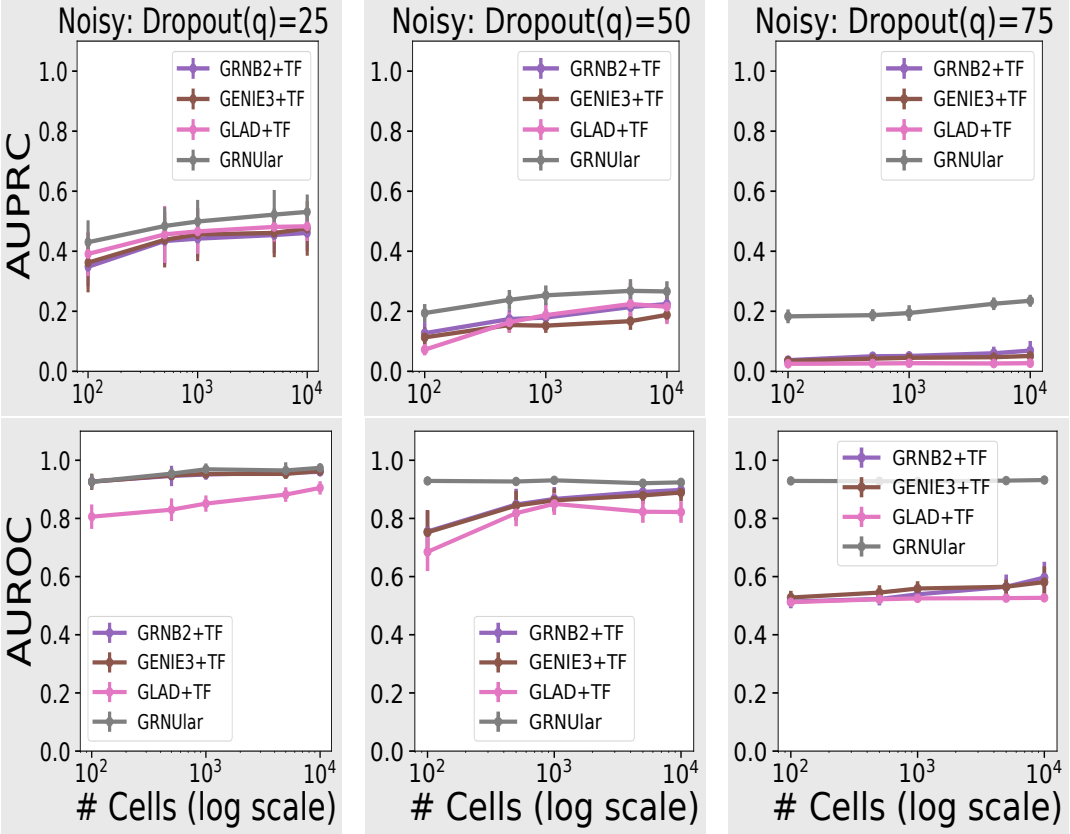

Figure 4.4: Noisy data setting of the SERGIO simulator with dropout-shape=20, D=100 and C=5. We vary the dropout percentile values as $[25, 50, 75]$ in both the upper panels (AUPRC values) and the lower panel (AUROC values). Larger q corresponds to higher technical noise. `GRNUlar` has a clear advantage in noisy settings.

Clean: simulated data with no technical noise

The 'clean' gene expression data from SERGIO follows all the underlying kinetic equations but excludes all the external technical noises. This data can be considered as being recorded with no technical errors. Fig. 4.3 compares different methods on their AUPRC performance on varying number of cells and number of cell types. For `GRNUlar` model we chose 2 layer NN with P=5, H_d={40, 60, 100}, L=15, and vary $\beta = \{2, 5\}$ in the loss function. The best model was chosen based on the validation results. We observe that `GRNUlar` consistently outperforms other methods.

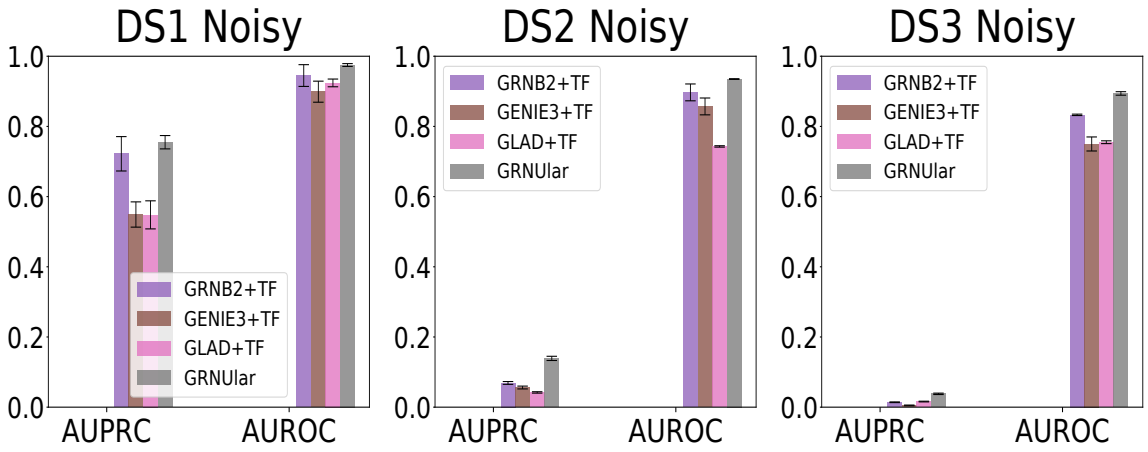

Figure 4.5: (Noisy settings, dropout percentile=82%) We report the average results over 15 test graphs in the noisy settings. `GRNUlar` gives notable AUPRC values and it outperforms other methods.

Noisy: simulated data with technical noise

We evaluate on the more challenging and realistic noisy settings. We limit varying the technical noise to dropouts while keeping the default settings for other SERGIO parameters. For higher levels of dropouts, researchers sometimes resort to data imputation techniques (which attempt to recover the number of molecules being dropped) as a preprocessing step which marginally improves results. For these experiments, we report results without the imputation preprocessing step and compare all methods directly on the noisy data obtained from the simulator. While training the models, we train on data with low dropout rates $q = \{0, 25\}$ and use the same models to predict networks on data with higher dropout rates. In Fig. 4.4, as we move towards the right, dropout percentile increases and we can observe deterioration in AUPRC values, although the `GRNUlar` model's AUROC performance is quite robust with increasing dropouts. Even in the case of 75% dropout value, where the other algorithms almost give output equivalent to random prediction, `GRNUlar` is able to handle this high percentage of missing information.

4.6.4 Realistic data from SERGIO: Ecoli & Yeast

The challenge for data-driven models is to be able to generalize to real datasets. Thus, it is important to test the ability of the unrolled algorithms to generalize over different settings from that of the training. To perform this study, we make use of the realistic data sets provided by the SERGIO simulator. They provide three scRNA-Seq datasets DS1, DS2 and DS3 which are generated from input GRNs with 100, 400 and 1200 genes respectively. These networks were sampled from real regulatory networks of *E. coli* and *S. cerevisae*. For each dataset, the settings are: number of cell types C = 9; total number of single cells M = 2700, and there are 300 cells per cell type. Each data set was synthesized in 15 replicates by re-executing SERGIO with identical parameters multiple times. The parameters were configured such that the statistical properties of these synthetic dataset are comparable to the mouse brain, given in [100].

We defined our training and testing settings such that there were considerable differences between them. We used all of the DS1, DS2 and DS3 datasets for testing. We train on data with settings similar to DS1, specifically the parameters like production cell rates, decays, noise-parameter and interaction strength. We trained with no dropouts as opposed to 82% dropout percentile in the case of the DS datasets. The datasets DS2, DS3 are completely different from training data (and DS1) in terms of the underlying GRN as well as the corresponding SERGIO parameters are sampled from different range of values. For details, refer to Table 1 and Appendix Tables S1, S3 in [25] for more insight on the differences in SERGIO parameter settings.

`GRNUlar` settings: We used a 2 layer neural network in the `fitDNN-fast` function for these experiments with a single hidden layer H_1. Following our strategy to choose the dimensions, $H_1 \sim 4 \times$ TFs. The number of TFs in DS1/DS2/DS3 are 10/37/127, respectively. So, we chose $H_1 = 40/200/500$ respectively, as the hidden layer dimensions. The other parameter settings remain similar to the ones mentioned in the previous subsections. Fig. 4.5 shows that `GRNUlar` performs better on these realistic datasets.

Table 4.1: Details of expression data from the BEELINE framework. To total number of genes for each data is 500 (highest varying genes)

Data	# TFs	# Expts
mDC	42	383
mESC	100	421
mHSC-E	69	1071
mHSC-GM	74	889
mHSC-L	83	847
hESC	86	758
hHep	56	425

4.6.5 Real single cell RNA-Seq datasets

We evaluated 21 gene expression datasets from the human and mouse species and their corresponding ground-truth networks [23].We evaluated different methods on seven datasets from five experiments which include human mature hepatocytes (hHEP) [101] , human embryonic stem cells (hESC) [102], mouse embryonic stem cells (mESC) [103], mouse dendritic cells (mDC) [104], and three lineages of mouse hematopoietic stem cells [105]: erythroid lineage (mHSC-E), granulocyte-macrophage lineage (mHSC-GM) and lymphoid lineage (mHSC-L). These are the same datasets used in [23] and we use their corresponding ground-truth networks for our experiments as well.

For each dataset there are three versions of ground-truth networks: cell-type-specific ChIP–seq, nonspecific ChIP–seq and functional interaction networks collected from STRING. We then have in all 21 different data pairs, 7 different types of expression data evaluated against 3 different types of ground truth.

Preprocessing the real data: For each gene expression data and its corresponding network, we first sorted all the genes according to their variance and select the top 500 varying genes. From the list of known TFs, we only considered all the TFs whose variance had p-value at most 0.01. We then found the intersection between the top 500 varying genes and all the TFs to get a subset of genes which act as the TFs, see Table 4.1. Then, we selected the sub-graph of top 500 varying genes from the underlying GRN as our ground truth for evaluation.

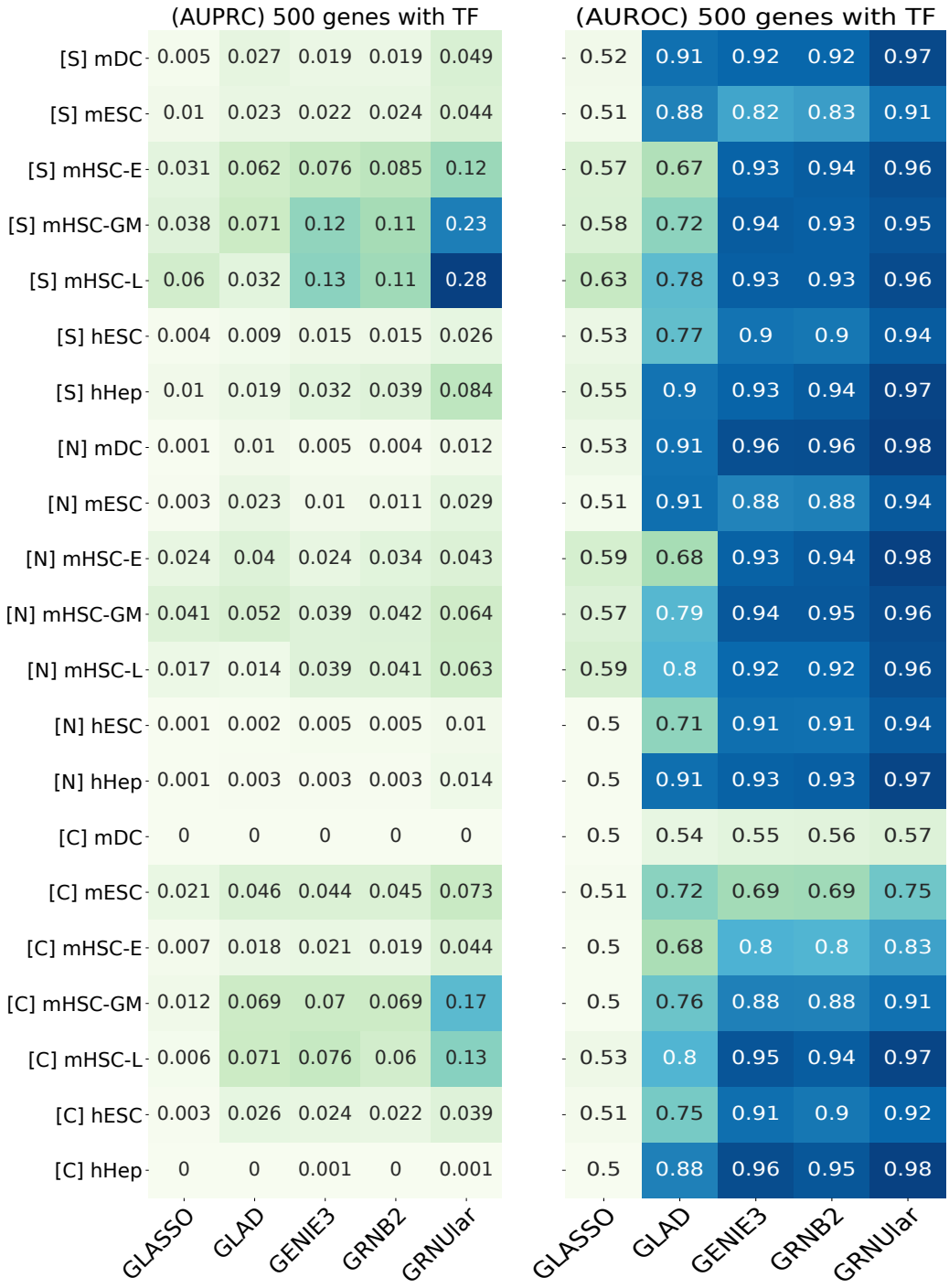

Figure 4.6: Heatmap of AUPRC and AUROC values of the real data from the BEELINE framework by [23]. We ran all the methods including the TF information. [S]/[N]/[C] represent the ground truth networks [String-network]/[Non-Specific-ChIP-seq-network]/[Cell-type-specific-ChIP-seq] respectively. Data of the species [m] mouse and [h] human were used. `GRNUlar` performs better than the other algorithms in both the metrics.

Training details: We train on the expression data which is similar to the SERGIO settings for the DS2 dataset as it has similar number of genes as the real data. We chose the underlying GRNs for supervision as the random graphs described in Section 4.6.3. We fixed the number of genes D = 400, the number of cell types C = 9 and total number of single cells M = 2700. The `GRNUlar` settings were $P = \{2, 5, 10\}, L = 15, H_d = \{200\}$ with two layer neural network.

In general for real data, we observed very low AUPRC values (refer Fig. 4.6); this is primarily due to the highly skewed ratio between true edges and total possible edges [74, 106]. The `GRNUlar` algorithm clearly outperformed other methods in all test settings. We can further improve the results by tuning the SERGIO simulator settings closer to the real data under consideration. Section 4.6.7 also compares the inference runtimes for various methods.

4.6.6 Analyzing the mESC network predicted by `GRNUlar`

We analysed the network predicted by `GRNUlar` from the mouse embryonic stem cell mESC dataset [103]. We chose TFs and genes corresponding to gene ontology (GO) terms related to ESC cell differentiation and cell fate towards endodermal cells as in this dataset the ESC cells are induced to differentiate into primitive endoderm cells [103]. From BioMart [107] we obtained 286 genes. We took the intersection between these genes with our predicted GRN (with 500 genes) and found 32 genes.

We first compared the interaction scores predicted by `GRNUlar` among all 500 genes and the scores among the 32 genes, without applying any threshold. We found that the latter set of scores is significantly higher than the former set of scores (see Fig. 4.7). This means there are more regulatory activities among the genes related to the expected biological processes compared to all the genes selected by variation. We then set the threshold for interaction score as 0.22, and obtained the network shown in Fig. 4.9. In this network, the TFs SOX7, SOX17, MTF2, GATA6 and CITED2 are known TFs in either stem cell

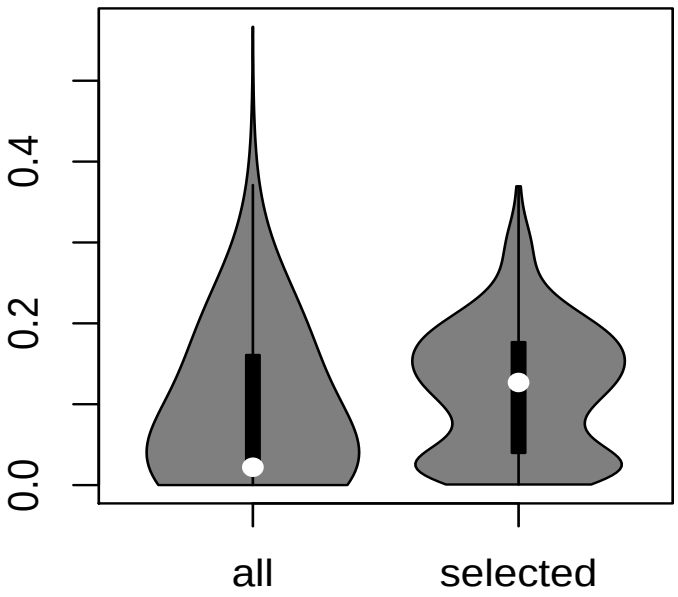

Figure 4.7: Violin plot comparing the scores of all interactions in the 500 genes (left) and scores of interactions between the 32 genes (right). Wilcoxon p-value is 1.3e-14.

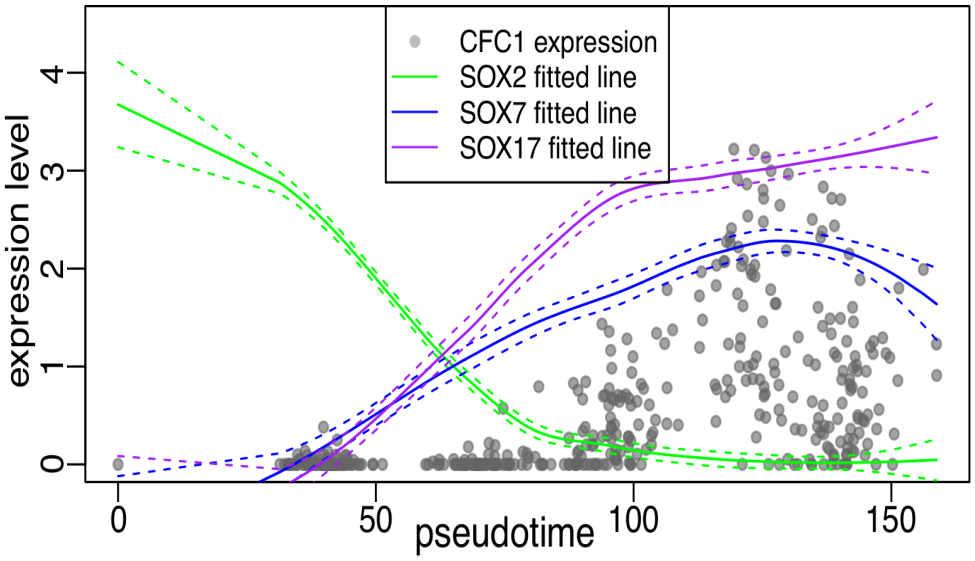

Figure 4.8: Comparison of gene-expression patterns over the pseudotime for CFC1 and the SOX family TFs.

differentiation or embryo development; NOTCH1 and RBPJ are TFs in the NOTCH pathway which controls cell fate specification (`www.genecards.org`). The TFs with highest interaction scores are highly relevant TFs for the cells under study.

We now show how our predicted interactions may bring new biological insights. For instance, we noticed that one of the target genes of SOX7 with strong interaction is CFC1.

From ChIP-Seq experiments (the [Cell-type-specific-ChIP-seq] ground truth network mentioned previously), SOX2 is a TF for CFC1.

However, we predicted SOX7 and SOX17 as the TFs for CFC1 in our results. We note that the dataset consists of ESC cells differentiating into primitive endoderm cells, and SOX2 is a key TF in mouse ESCs governing the pluripotency of the cells [108]. As the cells differentiate, the pluripotency goes down, so the SOX2 function may also decrease. To verify this, we use the pseudotime of the cells obtained from [23], which was inferred with Slingshot [109], and visualize the gene-expression levels of CFC1, SOX2, SOX7 and SOX17 (Fig. 4.8). For readability we plot the actual gene-expression levels cell by cell only for CFC1, and for the SOX TFs we plot the fitted lines of their expression levels obtained using LOESS regression. The dashed lines represent the standard deviation.

We see that indeed the SOX2 expression decreases along the pseudotime, and the expression levels of CFC1, SOX7 and SOX17 increase. The fitted lines of SOX7 and SOX17 show that they are much better predictors for the expression of CFC1 than SOX2. Indeed, it is discussed that SOX7 and SOX17 are highly related members of the SOX family and their high expression in ESCs are correlated with a downregulation of the pluripotency and an upregulation of the primitive endoderm-associated program [110]. This example showcases how we can use predicted regulatory networks to find regulatory pathways for a specific biological program. Some of these may already have evidence in literature but some may be new and our prediction can be used to provide hypothesis for further experimental validation.

4.6.7 Runtimes of different methods

Tables 4.2 & 4.3 show the inference time required for different methods with the TF information included. We run different methods on different platforms and hence comparing them directly is not fair. Although, we include them for the sake of completeness and to provide estimates of the run-times to the reader.

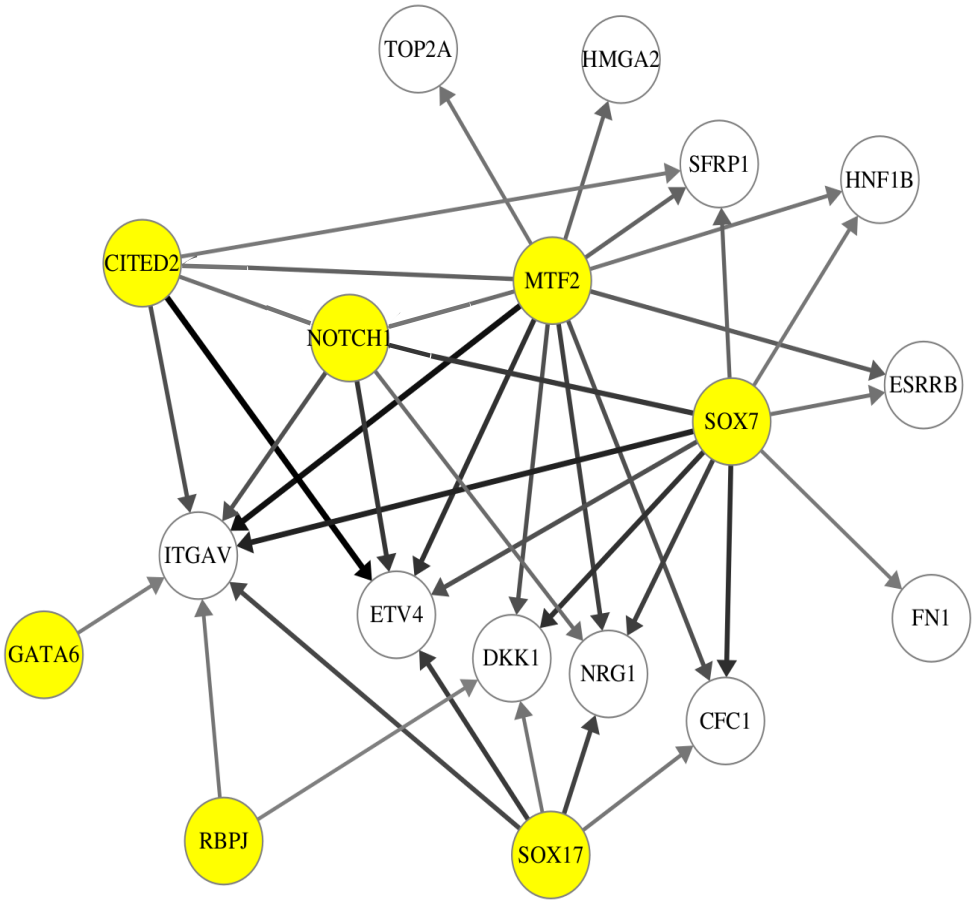

Figure 4.9: A subnetwork (CPDAG) with genes related to stem cell differentiation from `GRNUlar` predicted network. TFs are the nodes with yellow background. Darker edges mean higher predicted score for the interaction.

Table 4.2: Inference runtimes for the `GRNUlar` model with 2 layer NN, as we vary the hidden layer dimensions H_d. The time is reported for one complete forward call (`goodINIT` and `fitDNN-fast`) for D=1200 genes graph. Other relevant parameters settings were P = 5, L = 15, DNN epochs E1 = 400.

Time (secs)	H_d=200	H_d=500	H_d=1000
GRNUlar [gpu]	0.89	1.33	2.10

Table 4.3: Inference times for different methods on D = 1200 genes graph. The unrolled algorithms were ran on GPUs (NVIDIA P100s) while the traditional methods were ran on CPU having a single node with 28 cores.

Methods	GLASSO [cpu]	GRNB2 [cpu]	GENIE3 [cpu]	GLAD [gpu]	GRNUlar [gpu]
Time (secs)	180	612	1020	0.79	1.33

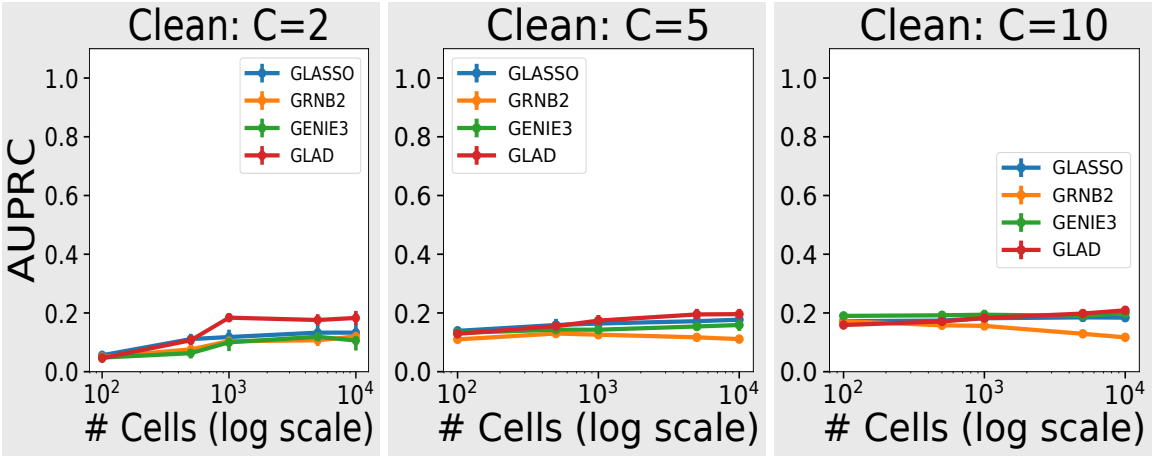

Figure 4.10: Clean data setting of the SERGIO simulator with D=100 genes. As the number of cell types increase from C=2 to C=10, we see that the AUPRC values increase in general. All the methods are ran without using the TF information. The unrolled algorithm `GLAD` in general outperform the traditional methods.

4.7 A case study of unrolled algorithms when TFs are absent.

In the settings where we do not have access to the transcription factors, we cannot use `GRNUlar` in these cases, but the unrolled algorithm `GLAD` can still be used. We found that `GLAD` consistently performs better than the traditional approaches. Here, we show our experiments that support our claim. This further highlights the potential of using unrolled algorithms for GRN recovery.

4.7.1 Experiments: On the clean and noisy settings of SERGIO

Clean: simulated data with no technical noise

We observe that `GLAD` works better than `GRNBoost2` in settings where TFs are absent, refer Fig. 4.10. It is often challenging for algorithms to perform well without the prior knowledge of Transcription factors and that can be observed in the experiments as well.

Noisy: simulated data with technical noise

Fig. 4.11 shows the performance of the methods without the TF information. We can see that as the dropout percentile increases, all the methods start to fail significantly. For example,

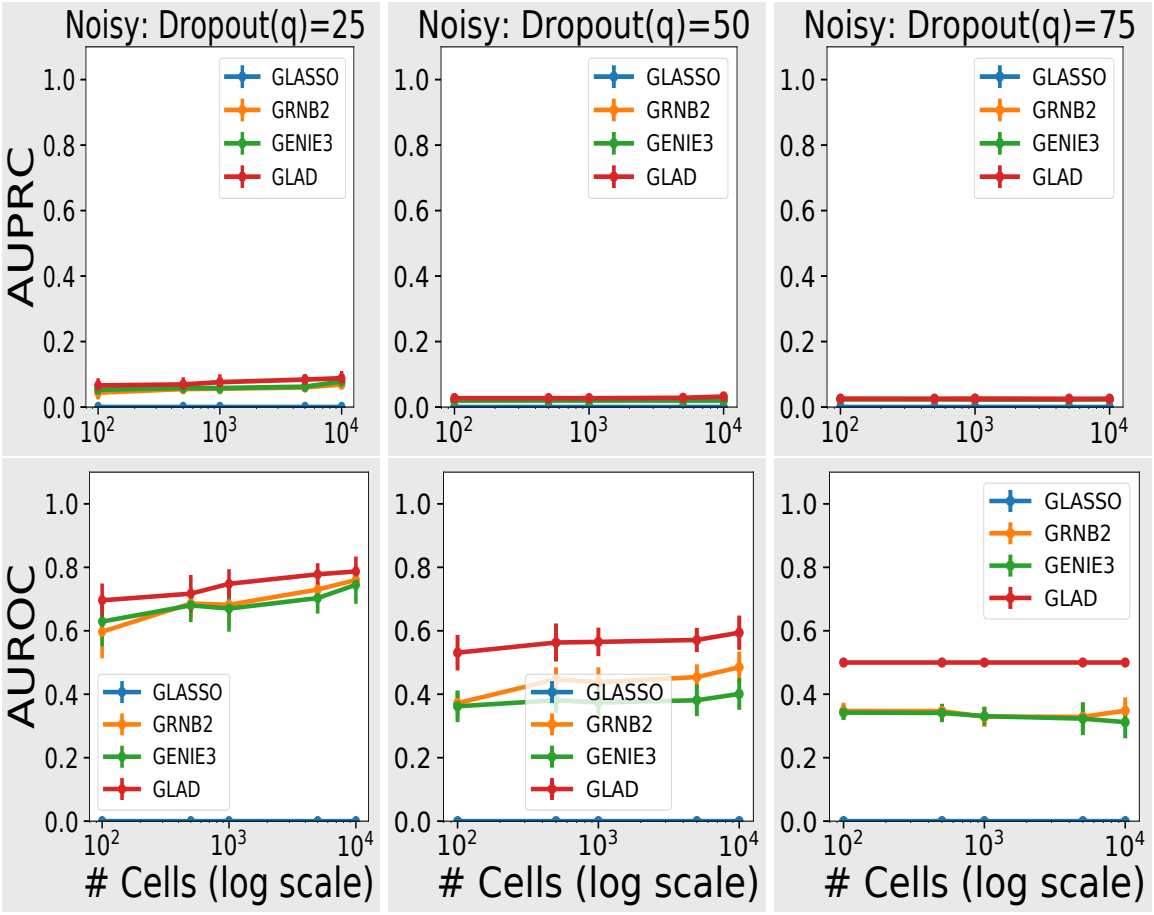

Figure 4.11: Without TF information : Noisy data setting with D=100 and C=5. We vary the dropout percentile values as $[25, 50, 75]$ in both the upper panels (AUPRC values) and the lower panel (AUROC values). Again, in the case of no TF, GLAD outperforms other methods.

the system becomes too ill-conditioned for the graphical lasso method. The cases where we do not provide the TF information, the GLAD method outperform other traditional methods.

4.7.2 Realistic data from SERGIO: Ecoli & Yeast

We also replicate the experiment settings for the realistic data setting for Ecoli and Yeast, given by the the SERGIO simulator. We again find that GLAD is able to outperform others for the realistic data settings as well. The results can be further improve if we find settings of the simulator that better resembles the test data under consideration.

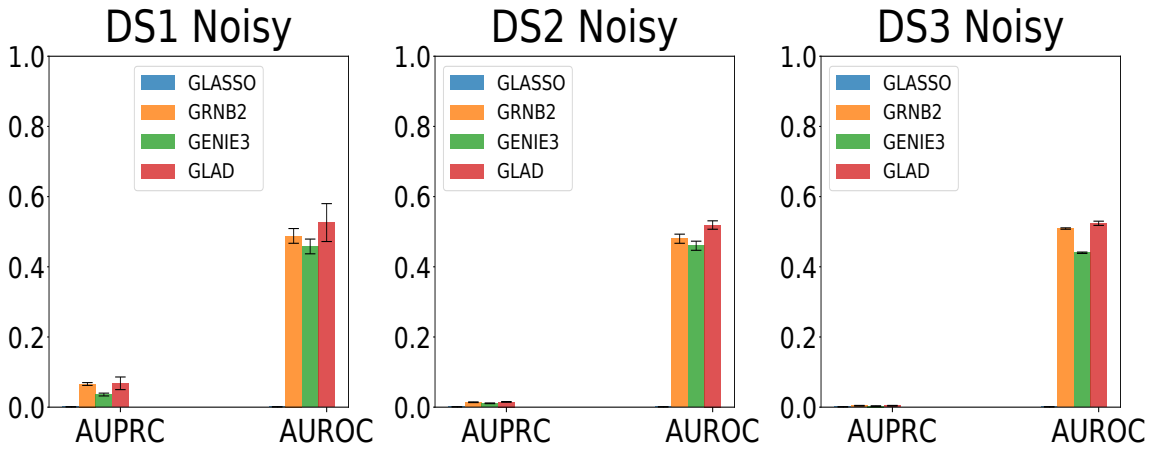

Figure 4.12: Noisy settings, dropout percentile=82% : We report the average results over 15 test graphs in the noisy settings. `GLAD` is the best performing method in absence of TFs.

4.8 Conclusions & Discussions

We present a deep unrolled supervised learning framework `GRNUlar` for gene regulatory network inference from scRNA-Seq data. The `GRNUlar` model combines the expressive ability of neural networks to capture complex regulation dependencies that manifest in expression data with unrolled learning of sparse graphical models to effectively emulate sparsity of the regulatory networks observed in the real world. We demonstrate that `GRNUlar` consistently outperforms the representative best-in-class methods on both simulated and real datasets, especially in the more realistic case of added technical noise. Our deep learning framework accommodates nonlinearity of the regulatory relationships between TFs and genes, and demonstrates high tolerance to technical noise in data.

The unrolled algorithm we proposed is the first supervised deep learning method for gene regulatory network inference from scRNA-Seq data. Our model learns the characteristics of the underlying network through simulated data from GRN-guided simulators like SERGIO, and demonstrates the novel and effective use of these simulators in training deep learning models, apart from their traditional use in benchmarking computational methods. Similar techniques can be investigated for other analysis tasks on single-cell data such as clustering and trajectory inference [111], by using available realistic simulators for scRNA-Seq data [25,

112].

Acknowledgements

We thank Aditya Pratapa and T.M. Murali for sharing the gold standard networks for real data used in their paper [23]. This research was supported in part through research cyberinfrastructure resources and services provided by the Partnership for an Advanced Computing Environment (PACE)[113] at the Georgia Institute of Technology, Atlanta, Georgia, USA.

CHAPTER 5
CONCLUSIONS

In this dissertation, we highlight the importance of using the prior knowledge as an inductive bias for designing deep learning architectures. We pursue this paradigm of deep architecture design and develop generic approaches applicable to wide variety of domains. We demonstrate the performance improvement in terms of accuracy, training data needed, runtime of the model alongside having more interpretability of the architecture.

Specifically, we provide generic templates for deep architecture design that can leverage the domain specific inductive bias and in-turn perform better than their traditional counterparts. This thesis provides two such novel techniques, namely 'Cooperative Neural Networks' and problem dependent 'Unrolled Algorithms', for designing deep learning architectures. We applied these techniques to problems related to Natural Language processing, document sentiment analysis, doing sparse graph recovery for gene regulatory networks. We have also separately evaluated our methods on complex task of protein structure recovery, graphical lasso based time-series modeling for finance data and problems related to drug discovery and healthcare. Throughout our experiments, we found that our deep architectures require considerably less amount of data for training and runs significantly faster than other competing deep architectures. This is mainly attributed to the considerably less number of learnable neural network parameters needed for our techniques.

In Chapter 2, we introduced Cooperative neural networks (CoNN), that is a new theoretical approach for implementing learning systems which can exploit both prior insights about the independence structure of the problem domain and the universal approximation capability of deep neural networks. We make the theory concrete with an example, CoNN-sLDA, which has superior performance to both prior work based on the probabilistic graphical model LDA and generic deep networks. While we demonstrated the method on text clas-

sification using the structure of Latent Dirichlet Allocation, the approach provides a fully general methodology for computing factored embeddings using a set of highly expressive networks. We also show that CoNN models in general require considerably less number of neural network parameters and thus gives us a significant runtime boost. Cooperative neural networks thus expand the design space of deep learning machines in new and promising ways. The Cooperative neural networks can also be plugged in an existing deep architecture pipeline seamlessly and can potentially improve the performance for various tasks.

Chapter 3 presents a novel neural network, `GLAD`, for the sparse graph recovery problem based on an unrolled Alternating Minimization algorithm. We theoretically prove the linear convergence of AM algorithm as well as empirically show that learning can further improve the sparse graph recovery. The learned `GLAD` model is able to push the sample complexity limits thereby highlighting the potential of using algorithms as inductive biases for deep learning architectures. Further development of theory is needed to fully understand and realize the potential of this new direction. Finally, we tested out `GLAD` on a real data simulator and found that data-driven learning can also help generalize to different test graph settings and potentially perform well on non-Gaussian distributions.

Finally, the Chapter 4 is a follow up work of the Chapter 3, where we expand the problem dependent 'Unrolled algorithm' technique to real world data. As the problem formulation from graphical lasso to regression based approach, we followed the same steps of designing unrolled algorithm for the regression based approach but it lead to a very diverse looking architecture. Therein comes the beauty and power of our technique of designing architectures based on the inductive biases.

Specifically, we present a deep unrolled supervised learning framework `GRNUlar` for gene regulatory network inference from scRNA-Seq data. The `GRNUlar` model combines the expressive ability of neural networks to capture complex regulation dependencies that manifest in expression data with unrolled learning of sparse graphical models to effectively emulate sparsity of the regulatory networks observed in the real world. We demonstrate

that GRNUlar consistently outperforms the representative best-in-class methods on both simulated and real datasets, especially in the more realistic case of added technical noise. Our deep learning framework accommodates nonlinearity of the regulatory relationships between TFs and genes, and demonstrates high tolerance to technical noise in data. We also show the superior performance of the unrolled algorithm GLAD in absence of TFs. Our work demonstrates the successful use of the supervision from data simulators for doing GRN inference. Similar techniques can be investigated for other analysis tasks on single-cell data such as clustering and trajectory inference, by using available realistic simulators for scRNA-Seq data.

The unrolled algorithms we proposed are the first supervised deep learning method for gene regulatory network inference from scRNA-Seq data. Our model learns the characteristics of the underlying network through simulated data from GRN-guided simulators like SERGIO, and demonstrates the novel and effective use of these simulators in training deep learning models, apart from their traditional use in benchmarking computational methods.

We believe that this dissertation has contributed substantially in providing proof of concept of a new paradigm of designing deep learning architectures, one that can reflect our belief of the underlying system of the domain under consideration. This work lies in the intersection of the fields of Deep learning, Probabilistic Graphical models and optimization with applications ranging from Natural Language processing, sparse graph recovery to Computational biology. We believe and hope that our work will be of interest to researchers having diverse backgrounds and bring value to society.

REFERENCES

[1] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[2] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[3] T. Mikolov, S. Kombrink, L. Burget, J. Černockỳ, and S. Khudanpur, "Extensions of recurrent neural network language model," in *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, IEEE, 2011, pp. 5528–5531.

[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[5] Q. V. Le *et al.*, "A tutorial on deep learning part 2: Autoencoders, convolutional neural networks and recurrent neural networks," *Google Brain*, pp. 1–20, 2015.

[6] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[8] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[9] A. Smola, A. Gretton, L. Song, and B. Schölkopf, "A hilbert space embedding for distributions," in *International Conference on Algorithmic Learning Theory*, Springer, 2007, pp. 13–31.

[10] D. Husmeier, R. Dybowski, and S. Roberts, *Probabilistic modeling in bioinformatics and medical informatics*. Springer Science & Business Media, 2006.

[11] J. Pearl, "Graphical models for probabilistic and causal reasoning," in *Quantified representation of uncertainty and imprecision*, Springer, 1998, pp. 367–389.

[12] P. Giudici and A. Spelta, "Graphical network models for international financial flows," *Journal of Business & Economic Statistics*, vol. 34, no. 1, pp. 128–138, 2016.

[13] M. Al Hasan and M. J. Zaki, "A survey of link prediction in social networks," in *Social network data analytics*, Springer, 2011, pp. 243–275.

[14] C. J. Needham, J. R. Bradford, A. J. Bulpitt, and D. R. Westhead, "A primer on learning in bayesian networks for computational biology," *PLoS Comput Biol*, vol. 3, no. 8, e129, 2007.

[15] N. Friedman, "Inferring cellular networks using probabilistic graphical models," *Science*, vol. 303, no. 5659, pp. 799–805, 2004.

[16] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[17] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski, *Robust optimization*. Princeton University Press, 2009, vol. 28.

[18] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[19] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM journal on imaging sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[20] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, Omnipress, 2010, pp. 399–406.

[21] J. Liu, X. Chen, Z. Wang, and W. Yin, "Alista: Analytic weights are as good as learned weights in lista," 2018.

[22] O. Banerjee, L. E. Ghaoui, and A. d'Aspremont, "Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data," *Journal of Machine learning research*, vol. 9, no. Mar, pp. 485–516, 2008.

[23] A. Pratapa, A. P. Jalihal, J. N. Law, A. Bharadwaj, and T. Murali, "Benchmarking algorithms for gene regulatory network inference from single-cell transcriptomic data," *Nature Methods*, pp. 1–8, 2020.

[24] T. Van den Bulcke, K. Van Leemput, B. Naudts, P. van Remortel, H. Ma, A. Verschoren, B. De Moor, and K. Marchal, "Syntren: A generator of synthetic gene expression data for design and analysis of structure learning algorithms," *BMC bioinformatics*, vol. 7, no. 1, p. 43, 2006.

[25] P. Dibaeinia and S. Sinha, "A single-cell expression simulator guided by gene regulatory networks," *bioRxiv*, p. 716 811, 2019.

[26] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *ECML*, 1998.

[27] J. D. Mcauliffe and D. M. Blei, "Supervised topic models," in *Advances in neural information processing systems*, 2008, pp. 121–128.

[28] W. Chong, D. Blei, and F.-F. Li, "Simultaneous image classification and annotation," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, IEEE, 2009, pp. 1903–1910.

[29] S. Lacoste-Julien, F. Sha, and M. I. Jordan, "Disclda: Discriminative learning for dimensionality reduction and classification," in *Advances in neural information processing systems*, 2009, pp. 897–904.

[30] J. Zhu, A. Ahmed, and E. P. Xing, "Medlda: Maximum margin supervised topic models for regression and classification," in *Proceedings of the 26th annual international conference on machine learning*, ACM, 2009, pp. 1257–1264.

[31] W. Li and A. McCallum, "Pachinko allocation:dag-structured mixture models of topic correlations," 2006.

[32] N. Srivastava, R. R. Salakhutdinov, and G. E. Hinton, "Modeling documents with deep boltzmann machines," *arXiv preprint arXiv:1309.6865*, 2013.

[33] H. Larochelle and S. Lauly, "A neural autoregressive topic model," in *Advances in Neural Information Processing Systems*, 2012, pp. 2708–2716.

[34] Y. Zheng, Y.-J. Zhang, and H. Larochelle, "A deep and autoregressive approach for topic modeling of multimodal data," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 6, pp. 1056–1069, 2016.

[35] Z. Yang, Z. Hu, R. Salakhutdinov, and T. Berg-Kirkpatrick, "Improved variational autoencoders for text modeling using dilated convolutions," *arXiv preprint arXiv:1702.08139*, 2017.

[36] Y. Miao, L. Yu, and P. Blunsom, "Neural variational inference for text processing," in *International Conference on Machine Learning*, 2016.

[37] Z. Gan, C. Chen, R. Henao, D. Carlson, and L. Carin, "Scalable deep poisson factor analysis for topic modeling," in *International Conference on Machine Learning*, 2015, pp. 1823–1832.

[38] Y. Tang and R. R. Salakhutdinov, "Learning stochastic feedforward neural networks," in *Advances in Neural Information Processing Systems*, 2013, pp. 530–538.

[39] Y. Bengio, E. Laufer, G. Alain, and J. Yosinski, "Deep generative stochastic networks trainable by backprop," in *International Conference on Machine Learning*, 2014, pp. 226–234.

[40] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," *arXiv preprint arXiv:1401.4082*, 2014.

[41] A. Mnih and K. Gregor, "Neural variational inference and learning in belief networks," *arXiv preprint arXiv:1402.0030*, 2014.

[42] Y. Kim, "Convolutional neural networks for sentence classification.," in *arXiv*, 2014.

[43] A. Dieng, "TopicRNN: A recurrent neural network with long-range semantic dependency.," in *arXiv preprint arXiv:1611.01702*, 2016.

[44] J. Chen, J. He, Y. Shen, L. Xiao, X. He, J. Gao, X. Song, and L. Deng, "End-to-end learning of lda by mirror-descent back propagation over a deep architecture," in *Advances in Neural Information Processing Systems*, 2015, pp. 1765–1773.

[45] J.-T. Chien and C.-H. Lee, "Deep unfolding for topic models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 2, pp. 318–331, 2018.

[46] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[47] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," in *NIPS*, 2015.

[48] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Advances in Neural Information Processing Systems*, 2017, pp. 3856–3866.

[49] H. S. Mousavi, T. Guo, and V. Monga, "Deep image super resolution via natural image priors," in *arxiv*, 2018.

[50] M. J. Wainwright, M. I. Jordan, *et al.*, "Graphical models, exponential families, and variational inference," *Foundations and Trends® in Machine Learning*, vol. 1, no. 1–2, pp. 1–305, 2008.

[51] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *International Conference on Machine Learning*, 2016, pp. 2702–2711.

[52] B. K. Sriperumbudur, A. Gretton, K. Fukumizu, G. Lanckriet, and B. Schölkopf, "Injective hilbert space embeddings of probability measures," 2008.

[53] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[54] J. Blitzer, M. Dredze, and F. Pereira, "Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification," in *Proceedings of the 45th annual meeting of the association of computational linguistics*, 2007, pp. 440–447.

[55] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[56] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[57] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky, "Tree-reweighted belief propagation algorithms and approximate ml estimation by pseudo-moment matching.," in *AISTATS*, 2003.

[58] E. Belilovsky, K. Kastner, G. Varoquaux, and M. B. Blaschko, "Learning to discover sparse graphical models," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 440–448.

[59] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems*, 2016, pp. 3981–3989.

[60] K. Li and J. Malik, "Learning to optimize," *arXiv preprint arXiv:1606.01885*, 2016.

[61] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 6348–6358.

[62] X. Chen, J. Liu, Z. Wang, and W. Yin, "Theoretical linear convergence of unfolded ista and its practical weights and thresholds," in *Advances in Neural Information Processing Systems*, 2018, pp. 9061–9071.

[63] J. Sun, H. Li, Z. Xu, *et al.*, "Deep admm-net for compressive sensing mri," in *Advances in neural information processing systems*, 2016, pp. 10–18.

[64] P. Ravikumar, M. J. Wainwright, G. Raskutti, B. Yu, *et al.*, "High-dimensional covariance estimation by minimizing l1-penalized log-determinant divergence," *Electronic Journal of Statistics*, vol. 5, pp. 935–980, 2011.

[65] B. Rolfs, B. Rajaratnam, D. Guillot, I. Wong, and A. Maleki, "Iterative thresholding algorithm for sparse inverse covariance estimation," in *Advances in Neural Information Processing Systems*, 2012, pp. 1574–1582.

[66] J. Friedman, T. Hastie, and R. Tibshirani, "Sparse inverse covariance estimation with the graphical lasso," *Biostatistics*, vol. 9, no. 3, pp. 432–441, 2008.

[67] Q. Sun, K. M. Tan, H. Liu, and T. Zhang, "Graphical nonconvex optimization via an adaptive convex relaxation," in *International Conference on Machine Learning*, 2018, pp. 4817–4824.

[68] A. J. Rothman, P. J. Bickel, E. Levina, J. Zhu, *et al.*, "Sparse permutation invariant covariance estimation," *Electronic Journal of Statistics*, vol. 2, pp. 494–515, 2008.

[69] R. Kannan and S. Vempala, "Randomized algorithms in numerical linear algebra," *Acta Numerica*, vol. 26, pp. 95–135, 2017.

[70] R. Mazumder and D. K. Agarwal, "A flexible, scalable and efficient algorithmic framework for primal graphical lasso," *arXiv preprint arXiv:1110.5508*, 2011.

[71] Z. Lu, "Adaptive first-order methods for general sparse inverse covariance selection," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 4, pp. 2000–2016, 2010.

[72] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[73] D. Marbach, J. C. Costello, R. Küffner, N. M. Vega, R. J. Prill, D. M. Camacho, K. R. Allison, A. Aderhold, R. Bonneau, Y. Chen, *et al.*, "Wisdom of crowds for robust gene network inference," *Nature methods*, vol. 9, no. 8, p. 796, 2012.

[74] S. Chen and J. C. Mar, "Evaluating methods of inferring gene regulatory networks highlights their lack of performance for single cell gene expression data," *BMC bioinformatics*, vol. 19, no. 1, p. 232, 2018.

[75] V. Y. Kiselev, T. S. Andrews, and M. Hemberg, "Challenges in unsupervised clustering of single-cell rna-seq data," *Nature Reviews Genetics*, vol. 20, no. 5, pp. 273–282, 2019.

[76] C. A. Vallejos, D. Risso, A. Scialdone, S. Dudoit, and J. C. Marioni, "Normalizing single-cell RNA sequencing data: Challenges and opportunities," *Nat. Methods*, vol. 14, no. 6, pp. 565–571, Jun. 2017.

[77] V. A. Huynh-Thu and G. Sanguinetti, "Combining tree-based and dynamical systems for the inference of gene regulatory networks," *Bioinformatics*, vol. 31, no. 10, pp. 1614–1622, 2015.

[78] A. Irrthum, L. Wehenkel, P. Geurts, *et al.*, "Inferring regulatory networks from expression data using tree-based methods," *PloS one*, vol. 5, no. 9, e12776, 2010.

[79] S. Kim, "Ppcor: An r package for a fast calculation to semi-partial correlation coefficients," *Communications for statistical applications and methods*, vol. 22, no. 6, p. 665, 2015.

[80] T. Moerman, S. Aibar Santos, C. Bravo González-Blas, J. Simm, Y. Moreau, J. Aerts, and S. Aerts, "Grnboost2 and arboreto: Efficient and scalable inference of gene regulatory networks," *Bioinformatics*, vol. 35, no. 12, pp. 2159–2161, 2019.

[81] H. Chen, J. Guo, S. K. Mishra, P. Robson, M. Niranjan, and J. Zheng, "Single-cell transcriptional analysis to uncover regulatory circuits driving cell fate decisions in early mouse development," *Bioinformatics*, vol. 31, no. 7, pp. 1060–1066, 2015.

[82] F. K. Hamey, S. Nestorowa, S. J. Kinston, D. G. Kent, N. K. Wilson, and B. Göttgens, "Reconstructing blood stem cell regulatory network models from single-cell molecular profiles," *Proceedings of the National Academy of Sciences*, vol. 114, no. 23, pp. 5822–5829, 2017.

[83] C. Y. Lim, H. Wang, S. Woodhouse, N. Piterman, L. Wernisch, J. Fisher, and B. Göttgens, "Btr: Training asynchronous boolean models using single-cell expression data," *BMC bioinformatics*, vol. 17, no. 1, p. 355, 2016.

[84] S. Woodhouse, N. Piterman, C. M. Wintersteiger, B. Göttgens, and J. Fisher, "Scns: A graphical tool for reconstructing executable regulatory networks from single-cell genomic data," *BMC systems biology*, vol. 12, no. 1, pp. 1–7, 2018.

[85] S. Aibar, C. B. González-Blas, T. Moerman, H. Imrichova, G. Hulselmans, F. Rambow, J.-C. Marine, P. Geurts, J. Aerts, J. van den Oord, *et al.*, "Scenic: Single-cell regulatory network inference and clustering," *Nature methods*, vol. 14, no. 11, pp. 1083–1086, 2017.

[86] T. E. Chan, M. P. Stumpf, and A. C. Babtie, "Gene regulatory network inference from single-cell data using multivariate information measures," *Cell systems*, vol. 5, no. 3, pp. 251–267, 2017.

[87] H. Matsumoto, H. Kiryu, C. Furusawa, M. S. Ko, S. B. Ko, N. Gouda, T. Hayashi, and I. Nikaido, "Scode: An efficient regulatory network inference algorithm from single-cell rna-seq during differentiation," *Bioinformatics*, vol. 33, no. 15, pp. 2314–2321, 2017.

[88] N. Papili Gao, S. M. Ud-Dean, O. Gandrillon, and R. Gunawan, "Sincerities: Inferring gene regulatory networks from time-stamped single cell transcriptional expression profiles," *Bioinformatics*, vol. 34, no. 2, pp. 258–266, 2018.

[89] M. Sanchez-Castillo, D. Blanco, I. M. Tienda-Luna, M. Carrion, and Y. Huang, "A bayesian framework for the inference of gene regulatory networks from time and pseudo-time series data," *Bioinformatics*, vol. 34, no. 6, pp. 964–970, 2018.

[90] A. T. Specht and J. Li, "Leap: Constructing gene co-expression networks for single-cell rna-sequencing data using pseudotime ordering," *Bioinformatics*, vol. 33, no. 5, pp. 764–766, 2017.

[91] A. I. Vân Anh Huynh-Thu, L. Wehenkel, and P. Geurts, "Inferring regulatory networks from expression data using tree-based methods," *PloS one*, vol. 5, no. 9, 2010.

[92] X. Chen, Y. Li, R. Umarov, X. Gao, and L. Song, "Rna secondary structure prediction by learning unrolled algorithms," *arXiv preprint arXiv:2002.05810*, 2020.

[93] D. M. Chickering, "Learning equivalence classes of bayesian-network structures," *Journal of machine learning research*, vol. 2, no. Feb, pp. 445–498, 2002.

[94] A.-C. Haury, F. Mordelet, P. Vera-Licona, and J.-P. Vert, "Tigress: Trustful inference of gene regulation using stability selection," *BMC systems biology*, vol. 6, no. 1, 2012.

[95] Z. Razaghi-Moghadam and Z. Nikoloski, "Supervised learning of gene-regulatory networks based on graph distance profiles of transcriptomics data," *NPJ systems biology and applications*, vol. 6, no. 1, pp. 1–8, 2020.

[96] Y. Yuan and Z. Bar-Joseph, "Deep learning for inferring gene relationships from single-cell expression data," *Proceedings of the National Academy of Sciences*, vol. 116, no. 52, pp. 27 151–27 158, 2019.

[97] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv preprint arXiv:1706.05098*, 2017.

[98] H. Shrivastava, X. Chen, B. Chen, G. Lan, S. Aluru, H. Liu, and L. Song, "{glad}: Learning sparse graph recovery," in *International Conference on Learning Representations*, 2020.

[99] S. Rajbhandari, H. Shrivastava, and Y. He, "Antman: Sparse low-rank compression to accelerate rnn inference," *arXiv preprint arXiv:1910.01740*, 2019.

[100] A. Zeisel, A. B. Muñoz-Manchado, S. Codeluppi, P. Lönnerberg, G. La Manno, A. Juréus, S. Marques, H. Munguba, L. He, C. Betsholtz, C. Rolny, G. Castelo-Branco, J. Hjerling-Leffler, and S. Linnarsson, "Brain structure. cell types in the mouse cortex and hippocampus revealed by single-cell RNA-seq," *Science*, vol. 347, no. 6226, pp. 1138–1142, Mar. 2015.

[101] J. G. Camp, K. Sekine, T. Gerber, H. Loeffler-Wirth, H. Binder, M. Gac, S. Kanton, J. Kageyama, G. Damm, D. Seehofer, *et al.*, "Multilineage communication regulates human liver bud development from pluripotency," *Nature*, vol. 546, no. 7659, pp. 533–538, 2017.

[102] L.-F. Chu, N. Leng, J. Zhang, Z. Hou, D. Mamott, D. T. Vereide, J. Choi, C. Kendziorski, R. Stewart, and J. A. Thomson, "Single-cell RNA-seq reveals novel regulators of human embryonic stem cell differentiation to definitive endoderm," *Genome Biol.*, vol. 17, no. 1, p. 173, Aug. 2016.

[103] T. Hayashi, H. Ozaki, Y. Sasagawa, M. Umeda, H. Danno, and I. Nikaido, "Single-cell full-length total RNA sequencing uncovers dynamics of recursive splicing and enhancer RNAs," *Nat. Commun.*, vol. 9, no. 1, p. 619, Feb. 2018.

[104] A. K. Shalek, R. Satija, J. Shuga, J. J. Trombetta, D. Gennert, D. Lu, P. Chen, R. S. Gertner, J. T. Gaublomme, N. Yosef, *et al.*, "Single-cell rna-seq reveals dynamic paracrine control of cellular variation," *Nature*, vol. 510, no. 7505, pp. 363–369, 2014.

[105] S. Nestorowa, F. K. Hamey, B. Pijuan Sala, E. Diamanti, M. Shepherd, E. Laurenti, N. K. Wilson, D. G. Kent, and B. Göttgens, "A single-cell resolution map of mouse hematopoietic stem and progenitor cell differentiation," *Blood, The Journal of the American Society of Hematology*, vol. 128, no. 8, e20–e31, 2016.

[106] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 233–240.

[107] R. J. Kinsella, A. Kähäri, S. Haider, J. Zamora, G. Proctor, G. Spudich, J. Almeida-King, D. Staines, P. Derwent, A. Kerhornou, *et al.*, "Ensembl biomarts: A hub for data retrieval across taxonomic space," *Database*, vol. 2011, 2011.

[108] S. Masui, Y. Nakatake, Y. Toyooka, D. Shimosato, R. Yagi, K. Takahashi, H. Okochi, A. Okuda, R. Matoba, A. A. Sharov, *et al.*, "Pluripotency governed by sox2 via regulation of oct3/4 expression in mouse embryonic stem cells," *Nature cell biology*, vol. 9, no. 6, pp. 625–635, 2007.

[109] K. Street, D. Risso, R. B. Fletcher, D. Das, J. Ngai, N. Yosef, E. Purdom, and S. Dudoit, "Slingshot: Cell lineage and pseudotime inference for single-cell transcriptomics," *BMC Genomics*, vol. 19, no. 1, p. 477, Jun. 2018.

[110] A. Sarkar and K. Hochedlinger, "The sox family of transcription factors: Versatile regulators of stem and progenitor cell fate," *Cell Stem Cell*, vol. 12, no. 1, pp. 15–30, Jan. 2013.

[111] M. D. Luecken and F. J. Theis, "Current best practices in single-cell RNA-seq analysis: A tutorial," *Mol. Syst. Biol.*, vol. 15, no. 6, e8746, Jun. 2019.

[112] X. Zhang, C. Xu, and N. Yosef, "Simulating multiple faceted variability in single cell rna sequencing," *Nature communications*, vol. 10, no. 1, p. 2611, 2019.

[113] PACE, *Partnership for an Advanced Computing Environment (PACE)*, 2017.

VITA

Harsh Shrivastava is a Ph.D. in ML candidate at the Computational Science & Engineering department at Georgia Tech. His research interests are at the intersection of Machine Learning, Probabilistic Graphical models and Convex & Non-linear optimization. He has publications in top Machine learning conferences like Neurips, ICLR, AAAI as well as reputed computational biology conferences like RECOMB, BIBM and holds 4 US patents. He has done research internships at Google Research & Google Brain (2020), Google X (2019), Microsoft Research Redmond (2018) and at Xerox PARC (2017). Prior to starting his Ph.D., he worked as a budding scientist at Xerox Research Center with focus on developing machine learning algorithms for healthcare. He received his B.Tech + M.Tech (dual degree) from Indian Institute of Technology Kharagpur in Electronics & ECE with specialization in VLSI engineering (2014). Harsh was born in the city of Indore, India and has stayed in various parts of India over the years.